Un Modèle Topologique pour le Raisonnement Diagrammatique

Mémoire, DEA de Sciences Cognitives, LIMSI,
Université Paris-XI, Orsay

Erika Valencia

Avril - Août 1997

Un Modèle Topologique pour le Raisonnement Diagrammatique

Mémoire, DEA de Sciences Cognitives, LIMSI,
Université Paris-XI, Orsay

Erika Valencia

Avril - Août 1997

Résumé

Dans le cadre des raisonnements de nature spatiale, nous introduisons une distinction entre les représentations et opérations qui font appel à une métrique, de celles qui s'en dispensent et qui sont topologiques. Le raisonnement diagrammatique désigne alors les raisonnements spatiaux qui ne font pas intervenir de métrique. Notre travail consiste à proposer une modélisation et des outils pour des raisonnements diagrammatiques.

Nous utilisons des concepts qui généralisent la notion de graphe, issus de la topologie algébrique, pour la représentation de situations diagrammatiques. A partir de cet outil, nous envisageons le raisonnement diagrammatique comme des opérations topologiques opérées sur une représentation à base de complexes simpliciaux.

Notre travail se concrétise par le développement de deux applications. L'une d'elles est la construction de représentations par classification, l'autre correspond à une tâche de raisonnement par analogies. Cette dernière application a donné lieu au développement du système **ESQIMO** en `Mathematica` pour la résolution de devinettes graphiques par des méthodes diagrammatiques.

Mots clés : Représentation des connaissances, Raisonnement diagrammatique, Topologie algébrique, Complexe simplicial abstrait, Analogie, Catégorisation, Modèles mentaux.

Remerciements

Le travail présenté dans ce document est le fruit d'une étroite collaboration avec mes deux encadrants, Jean-Paul Sansonnet et Jean-Louis Giavitto. Ils m'ont sans cesse guidée par leurs intuitions, leurs analyses et leurs critiques, et ils m'ont avant tout, soutenue et portée tout au long de cette période qui fût très intense pour moi. Leur compétence et leur disponibilité méritent toute ma reconnaissance.

De manière générale, toute l'équipe 81/2 m'a beaucoup aidée et conseillée, et je tiens à remercier ici Olivier Michel et Dominique De Vito pour leur gentillesse et l'intérêt réel qu'ils portent à mon travail.

Merci également à tous les membres de l'équipe «archi» qui ont contribué, chacun à sa manière, à ce que ce travail se déroule dans un environnement très agréable. En particulier, merci à Vincent Branger, Pablo Bosch, Olivier Auguste et Franck Cappello.

Je remercie Jean-Vincent Segard, qui a effectué son stage de DEA dans la même équipe, pour son amabilité, sa fraîcheur, mais surtout pour les discussions de biologiste qu'il a provoquées lors de nos trop nombreuses pauses-café. Merci également à Gaël De Chalendar, pour avoir répondu avec patience à toutes mes questions concernant les graphes conceptuels.

Un grand merci à Etienne Fieux, de l'équipe Topologie, au Département de Mathématiques d'Orsay, qui a bien voulu nous éclairer sur certains points de la topologie algébrique.

Enfin, merci à tous ceux qui m'ont relue, écoutée et supportée pendant cette fin de stage, Akim, Philippe, Frédéric, François, Jérôme, Samuel et Nadine.

Introduction

Dans ce rapport, nous présentons les travaux menés pendant le stage du *DEA de Sciences Cognitives* du **LIMSI** à Orsay. Ce stage de 5 mois, a pris place au sein de l'équipe *Architectures Parallèles* au **LRI**, à Orsay également, sous la direction de Jean-Paul Sansonnet.

L'objectif de ce travail est d'explorer les possibilités offertes par la topologie pour représenter des connaissances de manière distribuée. Au cours de cette exploration, nous avons été amenés à étudier certains modes précis de raisonnement, en particulier, nous avons introduit la notion de raisonnement diagrammatique.

Un raisonnement peut être défini comme un enchaînement d'énoncés ou de représentations conduit en fonction d'un but. Ce but peut être la démonstration, l'explication, l'interprétation, la décision, la justification, la compréhension, etc. Dans le cadre des *raisonnements de nature spatiale*, nous distinguons entre les représentations et les opérations qui font appel à une métrique, et celles qui s'en dispensent et qui sont purement topologiques. Le *raisonnement diagrammatique* désigne alors les raisonnements spatiaux qui ne font pas intervenir de métrique. Ils incluent certaines formes de raisonnement visuel, mais nous verrons qu'il ne se restreignent pas à cette seule voie sensorielle.

À partir de cette définition, notre travail consiste à proposer un outil pour modéliser des raisonnements diagrammatiques. Les outils classiques pour la représentation des connaissances développés en Intelligence Artificielle (logiques, réseaux sémantiques) sont peu adaptés à la modélisation d'un raisonnement qui met en œuvre des relations spatiales (voisinage, continuité, bord). Par ailleurs, bien que les qualités relationnelles des réseaux sémantiques nous semblent propices à la représentation de situations diagrammatiques, nous pensons que leur caractère géométrique n'est pas suffisamment exploité. Nous proposons alors d'introduire des concepts qui généralisent la notion de graphe, issus de la topologie algébrique.

Les complexes simpliciaux sont des structures définies par la topologie algébrique combinatoire, afin de construire et de représenter des espaces. Ils sont plus particulièrement utilisés pour étudier les notions de chemins et de bords. R. Atkin a proposé l'utilisation des complexes pour la représentation de relations binaires, notamment en sciences sociales, et a introduit des outils pour analyser cette représentation : sa méthode se nomme la Q-analyse. Nous proposons une extension de cette Q-analyse afin de pouvoir représenter des ensembles de prédicats.

À partir de cet outil, nous envisageons le raisonnement diagrammatique comme des opérations topologiques opérées sur une représentation à base de complexes simpliciaux. Afin d'illustrer notre proposition nous avons étudié deux applications qui impliquent deux activités clés pour les sciences cognitives. L'une d'elles est la construction de représentations par classification, l'autre est fortement connectée au domaine du raisonnement par analogies.

Notre objectif, avec la première application, est limité à montrer qu'il est possible d'extraire à peu de frais une représentation diagrammatique à partir d'une procédure classique de catégorisation. La deuxième application est plus ambitieuse et tente de mettre en œuvre des notions de topologie algébrique pour résoudre par analogie certains test de QI : il s'agit de compléter une suite trois figures A, B, C, par une quatrième figure D, de sorte que D soit à C ce que B est à A. Cette application a donné lieu au développement du système **ESQIMO** en `Mathematica`.

Structure du rapport

Le premier chapitre pose le cadre de notre travail. Nous introduisons l'idée que certaines capacités cognitives dépendent de la capacité à se représenter et à manipuler des relations spatiales. Nous nous intéressons exclusivement aux relations non-métriques que nous qualifions de *diagrammatiques*. Notre objectif est de développer des outils effectifs pour modéliser et automatiser le raisonnement sur les différentes formes de relations spatiales non-métriques. Nous montrons que les outils classiques utilisés en Intelligence Artificielle sont peu adaptés et nous proposons d'utiliser des notions de la topologie algébrique combinatoire.

Le chapitre deux fournit les éléments nécessaires de topologie algébrique. Nous illustrons les définitions mathématiques par un module `Mathematica` qui implémente la notion de complexe simplicial abstrait. Les notions ainsi définies sont ensuite utilisées pour représenter des relations binaires, suivant la démarche de la Q-analyse. Nous étendons ce type de représentation en introduisant le codage par un complexe simplicial d'un ensemble de prédicats. Ces types de représentations seront utilisés dans les deux chapitres suivants.

Le chapitre trois a un objectif limité : ils s'agit de montrer qu'il est possible d'extraire à peu de frais un complexe simplicial à partir d'une procédure classique de catégorisation. La procédure que nous étudions est issue d'un modèle proposé par Holland et *al* qui est désormais classique. Le résultat du processus de catégorisation fournit un complexe simplicial qu'il est possible de représenter sous forme de treillis.

Le chapitre quatre présente le système ESQIMO. Ce système est développé pour automatiser la résolution de problèmes du type tests de Q.I., par la résolution d'une analogie. Le principe d'ESQIMO est de représenter la transformation d'une configuration en une autre par un *chemin* dans un certain espace. Le problème est alors de construire un autre chemin, similaire au premier, et qui part d'une configuration donnée. Nous avons utilisé une notion élémentaire de similitude de chemin, mais le chapitre se termine par la perspective d'utilisation d'outils plus élaborées de topologie algébrique. Bien que ESQIMO ait été utilisé pour résoudre des devinettes graphiques, sa démarche n'est absolument pas spécifique du domaine d'application graphique.

La conclusion nous permet d'évoquer, après le rappel du travail effectué, les perspectives ouvertes par ce travail de DEA. Outre l'approfondissement nécessaire des études que nous avons commencées, nous montrons que les concepts et les techniques d'analyse développés dans ce document peuvent s'appliquer au domaine de l'analyse et de la synthèse de systèmes logiciels et matériels par *design patterns*.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Problématique : un outil topologique pour le raisonnement diagrammatique

Nous exposons ici les motivations de notre travail. Pour cela, nous commençons par poser le cadre de notre étude, à savoir la représentation et le raisonnement diagrammatiques (RRD). Le raisonnement diagrammatique constitue un domaine que nous avons délimité à l'intérieur du domaine des représentations et des raisonnements spatiaux. Nous avons défini les RRD comme l'élaboration et la manipulation de relations spatiales non métriques.

Notre objectif est la modélisation des processus de raisonnement diagrammatique, et nous avons donc besoin d'un formalisme de représentation des connaissances adapté. Les outils existants en Intelligence Artificielle, ne nous satisfont pas pour des raisons inhérentes à la nature du raisonnement diagrammatique. Ainsi, nous sommes amenés à introduire un formalisme différent, bien que pas tout à fait nouveau, pour la modélisation des RRD. Le formalisme que nous proposons dans le chapitre suivant, est issu de la topologie algébrique combinatoire : ce sont les complexes simpliciaux.

1.1 La notion de représentation et de raisonnement diagrammatique

1.1.1 Des formes d'intelligence variées

De nombreuses approches et positions ont été développées afin de cerner et de définir l'intelligence. Il apparaît aujourd'hui clairement que celle-ci ne peut pas se réduire à une faculté unique, bien définie et clairement mesurable, mais qu'elle recouvre au contraire des capacités variées relativement indépendantes les unes des autres.

Ainsi, un test de Q.I. posera un certain nombre de questions afin de mettre en évidence les connaissances dont dispose un sujet : son vocabulaire, ses capacités arithmétiques, son aptitude à se rappeler une série de nombres, de mots ou d'autres symboles, sa capacité à saisir la ressemblance entre deux éléments, etc. Mais on aurait pu lui demander de traduire ou de résumer un texte, de s'orienter dans une ville, ou une forêt, d'assembler un groupe d'images, de dresser une carte, de reconnaître un visage.

Les capacités exhibées par un individu lors de ces activités participent toutes de l'intelligence, selon n'importe quelle définition raisonnable du terme. Cependant, il existe des corrélations entre ces aptitudes, nous indiquant qu'il est possible de regrouper certaines compétences en ce que H. Gardner [Gar97] appelle des « formes d'intelligence » . Cette classification permet de développer des outils adaptés à l'analyse et à l'étude des mécanismes propres à chaque forme d'intelligence, et de corréler ces capacités à des structures biologiques comme les aires corticales par exemple. H. Gardner propose huit critères permettant de caractériser une forme d'intelligence spécifique :

- l'existence de lésions cérébrales détruisant ou épargnant sélectivement une faculté ;
- l'existence d'idiots-savants ou de prodiges ;
- l'existence d'une ou plusieurs opérations ou mécanisme de base traitant de l'information et permettant de représenter la connaissance spécifique d'un domaine ;
- une histoire développementale particulière distincte ainsi qu'un ensemble définissable de performances expertes ;
- une plausibilité évolutionniste ;
- une confirmation provenant de la psychologie expérimentale ;
- une confirmation provenant de la psychométrie ;
- la possibilité d'encodage dans un système symbolique (c'est-à-dire un système de signification culturellement inventé comme le langage, la peinture, les mathématiques, la musique).

Parmi les formes d'intelligence que H. Gardner distingue, nous allons nous intéresser à l'intelligence spatiale, et à une forme particulière d'intelligence spatiale, à savoir, le raisonnement diagrammatique.

1.1.2 L'intelligence spatiale

L. Thurstone [Thu38] décrit l'intelligence spatiale comme :

- l'aptitude à reconnaître l'identité d'un objet vu sous des angles différents ;
- l'aptitude à imaginer un mouvement ou un déplacement interne des différents éléments d'une configuration ;
- l'aptitude à penser les relations spatiales.

H. Gardner, résume, lui, l'intelligence spatiale par « l'aptitude à reconnaître le même élément sous différents angles, l'aptitude à transformer ou à reconnaître une transformation d'un élément en un autre, la capacité à évoquer une image mentale et à la transformer ensuite, la capacité à produire une représentation graphique d'un information spatiale ».

Ces capacités semblent étroitement liées à la vision, et R. Arnheim parle par exemple de «pensée visuelle» [Arn69]. Mais de notre point de vue, l'intelligence spatiale ne dépend pas uniquement du canal visuel, et il semble arbitraire de la lier à une modalité sensorielle particulière.

En effet, par *espace*, nous comprenons la représentation simultanée d'une multiplicité. L'image visuelle offre une telle multiplicité, mais c'est aussi le cas pour les autres sens, et l'on peut parler d'image tactile, gustative.

La cécité d'Euler ne semble pas avoir stoppé le développement de ses capacités de raisonnement et de visualisation mentale géométriques. Plus généralement, les aveugles de naissance peuvent facilement reconnaître les formes géométriques qui leur sont présentées comme des reliefs, par exemple, ils reconnaissent des images miroir.

Une étude portant sur les aptitudes spatiales chez l'aveugle montre qu'un enfant de 4 ans, aveugle de naissance, est capable d'utiliser une carte tactile afin de trouver un objet dans une pièce. La compréhension du principe d'une carte, y compris de ses symboles arbitraires, et son utilisation pour se guider, ne dépend donc pas de la modalité visuelle.

C'est pourquoi nous définissons provisoirement, dans ce document, l'intelligence spatiale comme **la capacité à élaborer et mémoriser une représentation de nature spatiale, et à structurer et parcourir mentalement cet espace.** Il faut comprendre par « représentation de nature spatiale » , la représentation simultanée d'une configuration d'éléments. Cette configuration s'analyse à travers des « relations spatiales » comme :

- la contenance,
- la connexité,
- le voisinage,
- le bord,
- l'intérieur,
- la dimension,
- la séparabilité,
- la symétrie,
- l'obstruction,
- l'orientation,
- la position,
- la distance,
- la déformation,
- la courbure,
- la rotation,
- la taille,
- ...

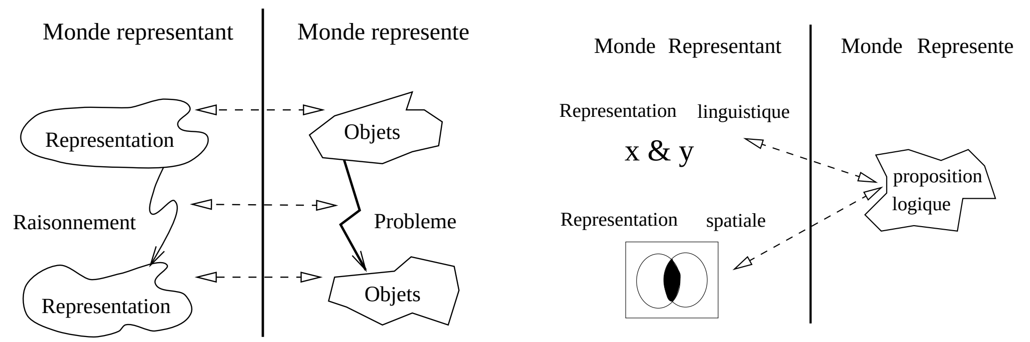
cette liste n'étant pas exhaustive.

1.1.3 Représentation et raisonnement spatiaux

Notre idée, qui n'est pas nouvelle, est que les relations spatiales permettent d'organiser et de structurer la représentation par *analogie* avec l'objet représenté. Une opération spatiale sur la représentation correspond alors à une opération ayant une sémantique naturelle dans le contexte de l'objet à représenter. Cette approche est illustrée par l'encart de la figure 1.1.

Nous pouvons, par exemple, développer une représentation spatiale de la logique du premier ordre. En effet, on peut mettre en correspondance une variable avec une région de l'espace, ce sont les diagrammes de Venn. Les opérations spatiales comme l'intersection ou l'union de ces régions possèdent une interprétation naturelle en logique du premier ordre, ce sont la conjonction et la disjonction des variables.

R. Arnheim va plus loin dans ce point de vue [Arn69] et soutient que les opérations cognitives viennent directement de notre perception du monde, la vision servant de système sensoriel paradigmatique. La table 1.1 présente quelques unes des analogies entre les opérations visuelles et des opérations cognitives abstraites développées par R. Arnheim.



(a) Monde représentant et monde représenté liés par un morphisme

(b) Représentation linguistique et représentation spatiale d'une proposition logique

FIG. **1.1** – Pour définir les représentations, on distingue entre deux mondes : le *monde représenté* et le *monde représentant*(figure 1.1(a)). Le monde représentant constitue la représentation du monde représenté. Ainsi, il comporte au moins une partie des croyances qu'un individu possède à propos du monde représenté [Ros95]. La représentation est différente de l'objet représenté tout en coexistant avec celui-ci. D'après Rossi [Ros95], seules les transformations mettant en œuvre une activité spécifique de codage aboutiraient à des représentations. Disons pour simplifier qu'un élément du monde serait en correspondance, un morphisme (représenté par les flèches en pointillés) avec un élément de la représentation, et qu'une tâche de résolution de problème dans le monde représenté correspond à un raisonnement dans le monde représentant. Les représentations peuvent être différentes selon les individus et les circonstances, pour un même objet. En effet, la nature de la représentation n'est pas supposée unique. Par exemple, au problème de savoir combien de carrés on obtient en pliant une feuille en deux, puis en deux, peut être résolu à l'aide d'une représentation spatiale, en comptant mentalement les pliures de cette image. Ou bien, on peut avoir recours à une représentation logico-mathématique, en multipliant 2 par 2. La sélection d'un type ou d'un autre de représentation dépendrait entre autres de la sélection de l'information à représenter. Celle-ci dépendant, à son tour, de la perception, du contexte, du codage, des connaissances antérieures. Ainsi, la représentation construite est indissociable du raisonnement qui lui sera appliqué.

1.1.4 La représentation et le raisonnement diagrammatiques (RRD)

Parmi les relations spatiales que nous avons citées, nous pouvons distinguer celles ayant un caractère purement *topologique* de celles qui ont un caractère *métrique*. De même, parmi les tâches de nature spatiale, nous pouvons distinguer les opérations *topologiques* et des opérations qui requièrent l'existence d'une *métrique*.

Dans ce travail, nous nous concentrons exclusivement sur les *représentations* et les *opérations spatiales topologiques*. Nous les qualifierons de *diagrammatiques*. Donnons des exemples d'opéra-

Opération cognitive	Image visuelle
Concept	Forme abstraite comme modèle de ce qui est vu dans l'image.
Attention sélective	Focus, fixation, accommodation.
Imaginer, deviner, résoudre	Voir le tout à partir de la partie, imaginer les parties cachées, reconstruire la forme à partir de ses projections.
Structurer, classer, apprendre	Voir les objets en relations, abstraction du contexte, présentation de l'objet isolé en le réduisant à ses invariants.

TAB. 1.1 – *Analogies entre des opérations visuelles et cognitives proposées par R. Arnheim*

tions et représentations diagrammatiques :

Carte symbolique. Une carte d'état major ressemble à l'objet qu'elle représente, à cause de ses caractéristiques métriques. Mais une carte des stations du métro parisien, est une représentation diagrammatique. En effet, établir un chemin entre deux stations est une tâche purement topologique qui ne s'intéresse pas à la métrique mais seulement à la connexité des segments de ligne empruntés.

Un autre exemple de carte symbolique est donné par la « carte du tendre », où la géographie d'un pays imaginaire reflète métaphoriquement l'organisation des concepts d'un domaine donné (ici, l'amour).

Les représentations graphiques d'objets logico-mathématiques. Plusieurs objets ou concepts mathématiques admettent une représentation spatiale ou géométrique. Les exemples sont assez connus pour ne pas les expliquer :

- diagrammes de Venn pour les formules logiques ;
- diagramme sagittal pour la représentation des relations ;
- graphe de Cayley pour les groupes ;
- représentation graphique d'une fonction réelle (cette représentation est diagrammatique en ce sens que, l'intérêt porte sur l'allure de la courbe, points asymptotiques, limites, points singuliers, croissance, décroissance, plutôt qu'à des valeurs exactes) ;
- les diagrammes de la théorie des catégories, une opération classique consistant à compléter le diagramme (preuve de l'existence d'un morphisme) ;
- représentation graphique des arbres (un arbre est un terme d'une algèbre) ;
- diagramme de Hasse pour la représentation d'un treillis.

Remarquons que les objets de cette liste n'ont pas une nature intrinsèquement spatiale. Mais nous pouvons a fortiori, ajouter tous les objets développés pour la formalisation de l'espace, à savoir, toute la topologie et la géométrie.

Diagrammes d'état en physique. Les diagrammes d'état en physique permettent de représenter spatialement le comportement de la matière. Par exemple, dans les diagrammes de changement de phase, les régions traversées par une courbe, indiquent les états successifs de la matière. Dans les diagrammes de Feynman, ce sont les interactions entre particules élémentaires qui sont représentées de manière spatiale.

Schémas fonctionnels. Les circuits électriques, les graphes data-flow, les graphes de Bond sont la représentation par des graphes de systèmes formés d'éléments fonctionnels.

Méthodes d'analyse. Plusieurs méthodes d'analyse et de conception de systèmes, tant matériel que logiciel ou organisationnel, font appel à des graphiques pour représenter des flux d'information et la structure du système à plusieurs échelles et selon des paramètres différents : par exemple, les organigrammes, actigrammes et datagrammes dans les méthodes SADT, Merise, etc.

Les jeux. Les jeux de plateau comme les Dames, les Échecs, le Go, sont décrits par des configurations spatiales de pièces, et l'arbre du jeu permet de rendre compte de la situation des coups.

Nous arrêtons ici les exemples de représentations et opérations diagrammatiques pour présenter des opérations spatiales qui ne sont pas diagrammatiques, car faisant appel à des propriétés métriques de l'espace :

- reconnaître la rotation d'une image-cible dans un ensemble d'images ;
- repérer l'immeuble le plus haut dans une ville ;
- déterminer le trajet le plus court pour sortir d'un labyrinthe ;
- piloter un avion.

1.2 Quels outils pour la représentation et le raisonnement diagrammatiques ?

Notre objectif est de développer des outils effectifs pour modéliser et automatiser la représentation et le raisonnement diagrammatiques.

Le raisonnement est étroitement lié au domaine de la résolution de problèmes, et raisonner nécessite l'examen de l'ensemble des solutions possibles dans un espace du problème. Nous pensons que cet espace du problème, en ce qui concerne les tâches de nature diagrammatique, doit posséder certaines propriétés particulières. En effet, les relations spatiales doivent être préservées et représentées dans un langage qui en facilite la manipulation.

Nous nous plaçons dans l'hypothèse, largement répandue aujourd'hui, d'une représentation distribuée des connaissances à manipuler. De plus, nous postulons que la résolution de problèmes par RRD doit procéder d'une résolution distribuée de plusieurs contraintes du problème. C'est pourquoi, nous parlons de résolution *parallèle*, et de représentation *distribuée*.

Ainsi, dans l'hypothèse d'un raisonnement basé sur des connaissances de nature diagrammatique, il faut postuler une représentation de type analogique *et* distribué, préservant les relations spatiales *et* temporelles, et permettant d'exprimer facilement la résolution en parallèle.

Nous allons rapidement présenter les outils développés en Intelligence Artificielle pour modéliser la représentation des connaissances, puis nous en ferons une critique avant d'esquisser notre proposition.

1.2.1 Outils pour la représentation des connaissances en Intelligence Artificielle

Nous n'avons pas l'intention de détailler ici l'ensemble des caractéristiques de chacun des outils existants en Intelligence Artificielle. Nous nous proposons de donner un bref aperçu des principales familles qui ont été développés pour la représentation des connaissances.

Les formalismes développés en IA, peuvent, pour la plupart, être qualifiés de symboliques. Parmi eux, nous distinguons quatre grandes familles :

- les logiques ;
- les réseaux sémantiques ;
- les primitives sémantiques ;
- les schémas.

G. Sabah nous expose un panorama détaillé des outils dans son livre [Sab90] en les envisageant sous l'angle de la Linguistique. Nous allons reprendre sa classification ici, même si nos préoccupations ne sont pas spécialement d'ordre linguistique.

Les outils logiques

Le caractère symbolique de tous les outils de l'IA, se justifie selon G. Sabah par le fait qu'un système d'intelligent doive utiliser un langage formel pour s'exprimer. Cela signifie alors que « toute expression de ce langage s'exprime à l'aide de symboles » [Sab90].

Ainsi, pour répondre à ces exigences d'ordre syntaxique, tout en fournissant des règles puissantes de raisonnement, un des premiers outils adoptés fût la logique des prédicats. L'adéquation de la logique des prédicats au traitement des langues soulevant un vaste débat, on développa ensuite des logiques dites « non classiques ». Parmi celles-ci, citons les *logiques modales* et les *logiques multivaluées*.

Les logiques modales permettent d'introduire des notions de causalité plus proches des notions manipulées par les humains. Elles permettent également d'attribuer des statuts différents à une même affirmation, en particulier, elles permettent d'introduire les logiques temporelles. Les logiques multivaluées ont l'avantage d'aller au-delà d'un codage binaire jugé trop caricatural pour ce qui concerne la modélisation du raisonnement humain.

Malgré leur grande efficacité et leur utilisation intensive en IA, des limites propres à la logique sont encore présentes. Citons par exemple, une trop grande nécessité de connaissance à priori dans le système pour rendre compte des phénomènes non exclusivement syntaxiques. De plus, «l'utilisation de règles formelles pour raisonner à propos de l'espace est combinatoirement trop complexe pour rendre compte de la facilité avec laquelle nous raisonnons sur de tels sujets» [JL92].

Les réseaux sémantiques

La famille des réseaux sémantiques prend son origine sur la base d'expériences en Psychologie. En partant de la constatation que notre aptitude à stocker et surtout à accéder à l'information semble obéir à un *principe d'économie*, Collins et Quillian [CQ69] posent la question de la structure de notre base de connaissances et font l'hypothèse d'une structure hiérarchique.

Ainsi, les réseaux sémantiques sont fondés sur la notion de *graphe*, les nœuds représentant des concepts et étant reliés par des arcs. Cette unité de concept n'acquiert tout son sens que par les relations qui les lient entre eux. Les nœuds comme les arcs sont étiquetés.

Ce modèle simple de représentation des connaissances a donné lieu à de multiples raffinements. Ainsi, en 1979, Fahlman propose les *réseaux à propagation de marqueurs* [Fah79] et Hendrix, les *réseaux partitionnés* [Hen79], Sowa propose en 1984 les *graphes conceptuels* [Sow84].

Les réseaux de Fahlman fonctionnent par propagation de marqueurs au niveau des nœuds. Cette propagation est effectuée en parallèle et l'information est transmise selon des conditions gérées par un contrôleur général.

Les réseaux partitionnés s'appuient sur la notion d'espace. Un espace est un sous-ensemble de nœuds et d'arcs qui peut être adressé comme un nœud. L'emboîtement de ces espaces traduit une hiérarchie. Il est ensuite possible d'extraire des vues locales ou globales sur ces ensembles d'espaces.

Les graphes conceptuels (GC), sont un modèle basé sur plusieurs sortes de graphes. Un treillis des types représente les types connus des objets de l'univers ainsi que les relations d'héritage entre eux. Des graphes dits canoniques représentent une configuration de concepts (petit réseau sémantique) qui sont élémentaires et utilisables à plusieurs endroits d'une représentation. Enfin, les graphes conceptuels sont construits à partir des graphes canoniques qui réfèrent à des types pour représenter une situation transitoire. Cette théorie constitue un véritable formalisme pour la manipulation des connaissances ainsi représentées. En particulier, on peut raisonner à l'aide de restrictions sur les types ou sur les référants, à l'aide de jonctions, de dérivations, de simplifications ou de projections de graphes.

De manière générale, les réseaux sémantiques reflètent une approche relationnelle et sont souvent présentés comme une alternative à la logique.

Les primitives sémantiques

Alors que le formalisme des réseaux sémantiques suppose que le *mot* est l'unité sémantique élémentaire, certains pensent que le sens peut être décomposé en éléments plus petits. Cette considération a mené vers des représentations *procédurales.* Ainsi, Schank [SA77] propose son modèle de *dépendance conceptuelle* Les concepts sont divisés en objets, actions et attributs. Dans un système procédural, les connaissances sont contenues dans des programmes qui savent comment agir dans des situations bien définies.

Les avantages d'une représentation *déclarative* sont : la flexibilité, la clarté et l'économie, mais les liens avec les processus de raisonnement sont implicites,tandis qu'une représentation procédurale permet une formulation directe entre les connaissances statiques et les processus de raisonnement.

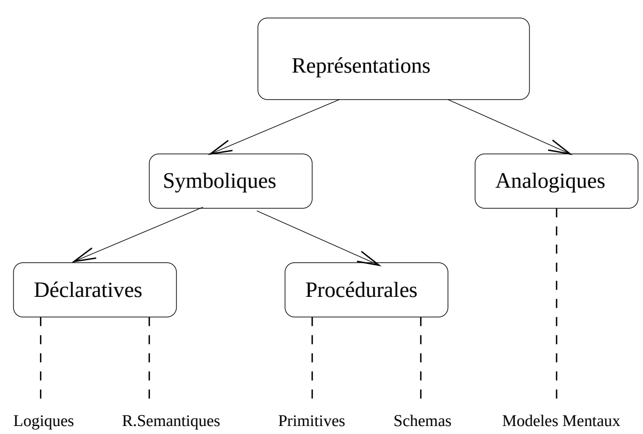
Les schémas

L'origine de ce modèle repose sur les observations de Minsky [Sab90]. Celui-ci essayait d'expliquer la rapidité avec laquelle nous semblons apercevoir d'un seul regard. Une explication qu'il propose est que lorsque quelqu'un s'apprête à entrer dans une pièce, il a déjà une idée à priori de ce qu'il peut y voir. Il propose alors d'associer un schéma à chaque nœud d'un réseau pour rendre compte de cette connaissance associés au concept. Les formalismes issus de ce modèle permettent de représenter des situations stéréotypées et typiques grâce aux *prototypes* ou des suites d´événements stéréotypés grâce aux *scénarios*. Ces connaissances sont également assorties d'aspects procéduraux.

L'idée contenue dans la notion de schéma étant que l'information sur l'environnement est stockée en mémoire par paquets qui sont accessibles en tant que tels et servent à produire des inférences et des solutions.

Classification

En définitive, si l'on veut reprendre les grandes lignes de la classification donnée par G. Sabah, nous pouvons séparer les représentations symboliques des représentations analogiques. Parmi les outils de nature symbolique, nous pouvons séparer ceux à caractère déclaratif de ceux à caractère plutôt procédural. La figure 1.2, montre cette classification.



TAB. 1.2 – *Classification des outils pour la représentation des connaissances en Intelligence Artificielle*

1.2.2 Critiques des outils classiques de l'IA du point de vue des RRD

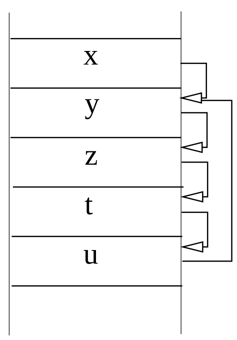
Nous venons de présenter rapidement les principales théories existantes en matière de représentation des connaissances. Nous avons vu que les outils développés en IA sont essentiellement symboliques.

Le choix d'un outil revient à choisir un formalisme dans lequel les caractéristiques du problème que l'on cherche à modéliser s'expriment le plus naturellement, facilement et efficacement possible. Or, les formalismes que nous venons de décrire conviennent mal à la représentation et à la manipulation de relations spatiales. Nous allons détailler nos critiques ci-dessous.

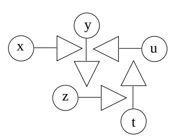
La logique. La logique a connu un grand succès dans les multiples applications développées pour l'IA en raison de son caractère constructif. En effet, elle revêt déjà un aspect symbolique et syntaxique tout à fait approprié à la programmation. De plus, elle est dotée de règles et de mécanismes de raisonnement que l'on a longtemps considérés comme proches du raisonnement humain.

Cependant, la modélisation de relations et d'opérations spatiales nous semble peu naturelle par le biais de la logique. En effet, prenons l'exemple du connecteur logique C, introduit initialement par Clarke [Cla81], pour caractériser qualitativement des propriétés et des relations spatiales à l'aide simplement de la notion de région. Ce connecteur a été développé et utilisé pour essayer de caractériser un tore de façon topologique par N. Gotts [Got94]. Celui-ci propose de développer une taxonomie des propriétés topologiques en utilisant le connecteur C, et ses travaux commençent par la définition d'un tore. Mais en écrivant son article, Gotts reconnaît que « d'exprimer des concepts intuitivement simples en termes d'une primitive et quelques axiomes n'est pas facile ». De plus, il déclare rencontrer des difficultés à modéliser les propriétés topologiques sans pouvoir référer aux liens conceptuels qui découlent habituellement entre les notions de dimension ou de bord. Il estime que le travail à produire est encore immense afin de pouvoir l'appliquer au raisonnement spatial et qualitatif (« spatial and qualitative reasonning » [Got94]).

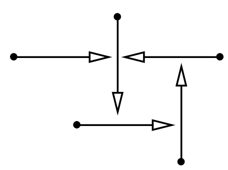
Les réseaux sémantiques. C'est surtout à la famille des réseaux sémantiques que nous nous intéressons. L'aspect relationnel des réseaux nous semble primordial. En effet, selon nous, c'est d'abord les liens entre les concepts qui les définissent. Nous avons illustré cette dernière remarque à l'aide de la métaphore informatique des pointeurs, figure 1.2.



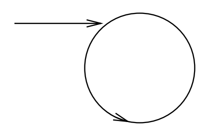
(a) Configuration de cinq pointeurs x, y, z, t, u, en mémoire



(b) Représentation sous forme de graphe des relations entre les cinq pointeurs



(c) Structure intrinsèque sous-jacente de la configuration des cinq pointeurs



(d) Idée, essence de la structure

FIG. **1.2** – En 1.2(a), on se donne une structure de cinq pointeurs en mémoire d'ordinateur nommés x, y, z, t, u. La figure 1.2(b), représente le graphe associé à cette configuration, il est indépendant des noms utilisés pour désigner les pointeurs. En 1.2(c), nous montrons que l'on peut même se passer de les nommer, les pointeurs sont alors vus comme des entités abstraites qui s'auto-réfèrent. Cet exemple illustre le rôle fondamental que jouent les relations dans la définition d'une structure.

Par ailleurs, les traitements proposés par les graphes conceptuels pour raisonner sur les représentations, ont un caractère très intuitif et visuel. En effet, une jonction de deux graphes, une simplification, une restriction, sont autant d'opérations qui admettent une interprétation spatiale.

P. Bernat a justement utilisé le formalisme des graphes conceptuels pour développer un logiciel d'aide au raisonnement géométrique [Ber97], appelé CHYPRE. Il a dû, pour cela, étendre quelque peu le formalisme pour une utilisation plus adaptée au raisonnement géométrique. Nous allons présenter son système, car c'est un exemple permettant de discuter de la capacité des réseaux sémantiques à supporter un raisonnement de nature diagrammatique.

La résolution d'un problème de géométrie doit passer, selon Bernat, par la représentation de l'espace du problème sous forme elle-même géométrique. En effet, des systèmes comme GT (Geometry Tutor [ABY85]), se focalisant uniquement sur la démonstration des problèmes de géométrie, sont basés sur un formalisme logique. Le raisonnement adopté lors d'une démonstration peut être de type déductif ou inductif, cependant, le raisonnement lors de la démonstration n'est pas à confondre avec le raisonnement nécessaire à la recherche d'une solution dans l'espace du problème.

Pour l'étape de recherche de la solution, il est plus judicieux, selon Bernat, d'emprunter un formalisme basé sur la géométrie. Cela permet en particulier d'intégrer des connaissances implicites, très lourdes à intégrer dans un système logique (par exemple, le milieu d'un segment). La démonstration de la solution trouvée est ensuite aisée car, selon Taurisson : « démontrer c'est voir derrière l'objet, la structure des relations » [Tau93].

Les adaptations nécessaires du formalisme des graphes conceptuels, pour l'application au raisonnement géométrique sont les suivantes :

- le référent unique est adapté pour pouvoir avoir plusieurs référents. En effet, un objet en géométrie possède plusieurs référents dans sa définition ;
- la définition d'une classe d'équivalence sur les étiquettes des objets, car deux objets géométriques définis dans un ordre différent peuvent être équivalents (par exemple les parallélogrammes $ABCD$ et $BCDA$).

Les caractéristiques des graphes conceptuels mises en avant pour le choix pour ce genre de modélisation sont aussi mentionnés, ce sont les suivantes :

- ce formalisme est un modèle de la perception ;
- les GC respectent les relations entre les composants ;
- le modèle est évolutif ;
- comme pour les objets mathématiques, les concepts des GC ne sont pas atomiques mais contiennent des relations internes. En effet, un objet de mathématiques contient dans sa définition toutes les définitions de ses composants.

Nous sommes d'accord avec Bernat sur ces points positifs offerts par les GC pour la modélisation du raisonnement diagrammatique. Cependant, nous pensons que ce ne sont pas les propriétés géométriques des réseaux qui sont mises en avant dans ce formalisme. De plus, plutôt que de définir une classe d'équivalence sur les étiquettes des objets, il faudrait choisir un formalisme qui n'induise pas cette distinction entre deux représentations d'un même objet. Cependant, nous voulons préserver les avantages offerts par une représentation de type *graphes*, tout en augmentant ce formalisme pour le rendre plus expressif dans les domaines de la représentation et le raisonnement diagrammatiques.

Enfin, nous sommes séduits par l'idée introduite dans les modèles mentaux que la représentation puisse être construire par *analogie* avec la nature du problème traité. Ainsi, nous chercherons à emprunter un formalisme qui soit du domaine de la géométrie.

L'approche des Modèles mentaux. Un modèle mental est une représentation interne d'un état des choses (state of affairs) du monde extérieur. Il s'agit d'une forme de représentation des connaissances reconnue par de nombreux chercheurs en sciences cognitives, comme étant la façon naturelle par laquelle l'esprit humain manipule la réalité. La théorie des modèles mentaux doit son origine à trois éminents penseurs : le philosophe L. Wittgenstein, le psychologue K. Craik et le chercheur en sciences cognitives D. Marr [JL92].

Il existe un degré minimal d'analogie dans tout modèle mental. Cependant, lorsque ce modèle est destiné, par exemple, à figurer les relations spatiales parmi un ensemble d'éléments, l'analogie franchit un degré supplémentaire, dès lors le modèle s'organise selon une structure dimensionnelle dans laquelle les distances sont préservées. L'image est alors une expression du modèle, tel que celui-ci se trouve saisi sous un certain point de vue [Den92].

Proposition : la topologie algébrique pour modéliser les RRD. La topologie nous fournit des outils qui vont nous permettre de faire le lien entre les graphes (des réseaux sémantiques), les relations spatiales et toutes les opérations de nature spatiale.

1.3 Outils topologiques pour les RRD

Les caractéristiques du raisonnement diagrammatique nous ont fait chercher parmi les outils existant en IA des aspects de nature fortement visuelle, intuitive, relationnelle et analogique. Dans la perspective d'une formalisation de ces phénomènes, nous proposons tout naturellement la topologie. La branche de la topologie sur laquelle nous allons travailler est la topologie algébrique et en particulier la topologie combinatoire, car celle-ci est à même d'offrir un support effectif pour développer des outils informatiques permettant d'automatiser la représentation et le raisonnement diagrammatiques. De manière synthétique, nous pouvons résumer nos influences par le schéma de la table 1.3.

1.3.1 Les avantages d'un modèle topologique

Toutes les notions intervenant dans le RD que nous voudrions modéliser seront facilement exprimables grâce à la topologie algébrique. En particulier les notions de :

- *espace* : relations spatiales et opérations géométriques ;
- *temps* : évolution des données, des événements ;
- *parallélisme* : application en parallèle de plusieurs processus de raisonnement classique;
- *distribution* : des connaissances à manipuler.

En effet, une structure en réseaux étant particulièrement bien adaptée pour modéliser une connaissance distribuée, un réseau multi-dimensionnel le sera d'autant plus.

L'espace, tout naturellement sera une notion adaptée à la topologie qui est une branche de la géométrie. Le temps, quant à lui, sera facilement accessible à la manière des physiciens comme une quatrième dimension ou encore d'une façon proche des logiques modales temporelles avec une relation d'accessibilité entre les mondes passés et futurs.

De plus, la topologie algébrique combinatoire est de nature discrète et nous permettra de représenter facilement des connaissances aussi bien symboliques qu'analogiques. Nous reverrons tous ces points d'une manière plus technique dans le chapitre suivant.

Enfin, les concepts et les théorèmes de la topologie combinatoire que nous pourrons appliquer sur cette structure nous semblent tout à fait adaptés pour modéliser les mécanismes qui entrent en jeu lors des processus de raisonnement diagrammatique.

En effet, nous verrons que les notions centrales de la topologie sont celles de transformation et de déformation ; ainsi les théorèmes que nous utiliserons ont une forte composante géométrique.

1.3.2 Applications de notre hypothèse, objectifs de notre travail

Notre hypothèse consiste à postuler une représentation de type analogique pour la représentation et le raisonnement diagrammatique. Nous allons donc la tester dans deux tâches importantes des sciences cognitives :

- l'élaboration de la représentation lors d'une extraction par classification ;
- l'utilisation de cette représentation lors d'une résolution de problème par raisonnement diagrammatique utilisant un raisonnement par analogie.

Extraction et représentation des connaissances

En nous basant sur l'illustration des modèles mentaux proposée par Holland [HHNT86], nous nous intéresserons tout d'abord aux interactions de l'individu avec son environnement. C'est l'étape de l'extraction des concepts à partir d'un flux de données.

Nous supposerons qu'un système cognitif représente le monde avec lequel il interagit à l'aide de modèles mentaux, et nous chercherons à préciser la nature de ce modèle.

Premièrement, un modèle doit permettre au système de produire des prédictions même si la connaissance de l'environnement est incomplète. Ensuite, il doit être facile à raffiner à mesure que de nouvelles informations sont acquises sans perdre l' information précédemment intégrée. Nous verrons qu'un système cognitif utilise pour cela la catégorisation en divisant l'environnement en classes d' équivalence.

Pour satisfaire ces conditions, tout en restant dans un cadre topologique, nous introduisons la notion mathématique de morphisme. Nous nous demanderons ensuite avec Gineste [Gin97] et Holland [HHNT86] si ce morphisme est un isomorphisme, un homomorphisme ou un quasi-homomorphisme.

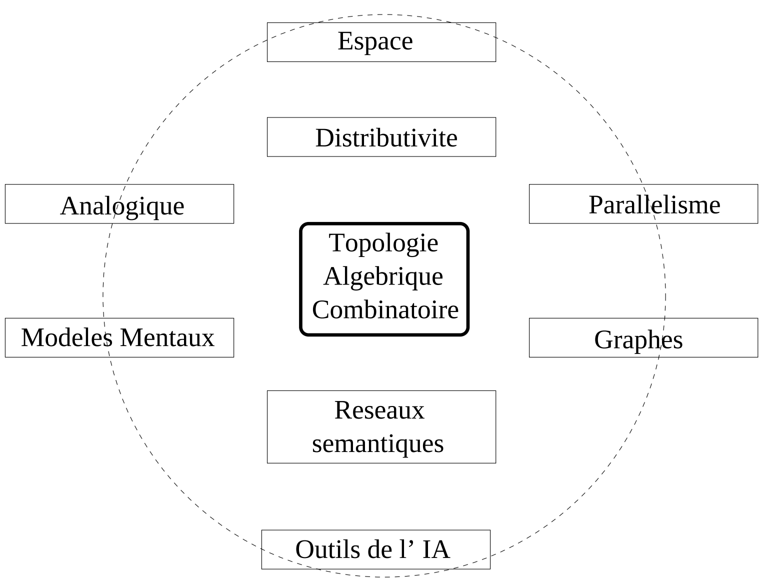
Une fois le monde partitionné en classes d'équivalence, nous proposons un moyen pour structurer ces connaissances en une ontologie à l'aide des outils de la topologie algébrique. Nous l'illustrerons à l'aide d'une petite simulation.

Nous supposerons que des concepts d'un niveau sémantique supérieur peuvent également être élaborés selon ce même schéma, et nous en proposerons aussi une petite illustration.

Modélisation d'un raisonnement diagrammatique par analogie

Dans un deuxième temps, nous ferons l'hypothèse de cette représentation sous sa forme topologique et nous étudierons la modélisation d'un raisonnement diagrammatique, à l'aide, justement, d'opérations qui nous sont offertes par la topologie.

Pour cela, nous étudierons la résolution de problèmes du type « tests de Q.I. » par résolution d'une analogie. Nous en proposerons une implémentation que nous illustrerons par des exemples commentés.



TAB. 1.3 – *Motivations et influences*

Chapitre 2

Éléments de topologie algébrique pour les RRD

Dans ce chapitre, nous commencerons par donner quelques idées intuitives sur l'objectif de la topologie et quelques unes de ses méthodes [1]. Nous introduirons ensuite plus formellement un concept élémentaire de topologie algébrique, le complexe simplicial abstrait. Nous donnerons alors une implémentation de ces notions en `Mathematica`.

Le reste du chapitre est consacré à l'utilisation de la notion de complexe simplicial pour la représentation des connaissances. Nous représenterons plus particulièrement les relations binaires et les ensembles de prédicats. La mise en œuvre des concepts et outils présentés dans ce chapitre sera faite dans les deux chapitres suivants.

2.1 Généralités sur la topologie

2.1.1 Approche intuitive des objectifs de la topologie

La topologie a pour objectif d'étudier l'*espace*. Un espace est un ensemble de points muni d'une relation de voisinage : dans la notion d'espace, on s'intéresse plus à la structure de voisinage qu'aux points pour eux-mêmes. Ainsi, on essaie de formaliser les notions de *continuité*, de *limite*, de *bord*, de *chemin*, etc.

Un concept important est celui de *déformation continue* : considérons un anneau A découpé dans une feuille de caoutchouc et déformons-le mentalement. Nous pouvons le tordre, le tirer, le comprimer, toutes les déformations qui n'impliquent pas de déchirure ou de collage sont permises pour obtenir un nouvel objet B. Ces déformations préservent la structure de l'espace dans le sens où la notion de voisinage est préservée dans la transformation de A en B ; autrement dit, les voisins d' un point de A sont transformés en voisins de la transformée du point.

Par exemple, considérons l'anneau de la figure 2.1(a), il est possible de le déformer continûment pour obtenir une poule 2-dimensionnelle dont l'unique trou soit son œil, comme le montre la figure 2.1(b). De même, si nous considérons une bouée, c'est à dire un tore 2.2(a), nous pouvons la déformer continûment en une tasse comme le montre la figure 2.2(b) [Got94]. En effet, le trou de la bouée est conservé puisqu'il constitue l'anse de la tasse, tout le reste étant réparti sans trou sur le reste de la tasse.

Nous dirons que l'anneau et la poule de notre dessin sont *topologiquement équivalents*, de même pour la bouée et la tasse. Les propriétés de l'anneau disque plein qui nous intéressent ici sont qu'il

1. Les références suivantes ont été utilisées pour les généralités sur la topologie et la topologie algébrique : [RS97, Lak84, Die86]. Les ouvrages suivants constituent des références connues sur la topologie algébrique et permettent de s'initier au sujet : [Ale82, Hen94, HY88, God71]. L'utilisation de la topologie algébrique pour la modélisation, en particulier en sciences sociales, a été introduite par [Atk74, Atk77], les références les plus accessibles étant sans doute [Cas92, Cas94].

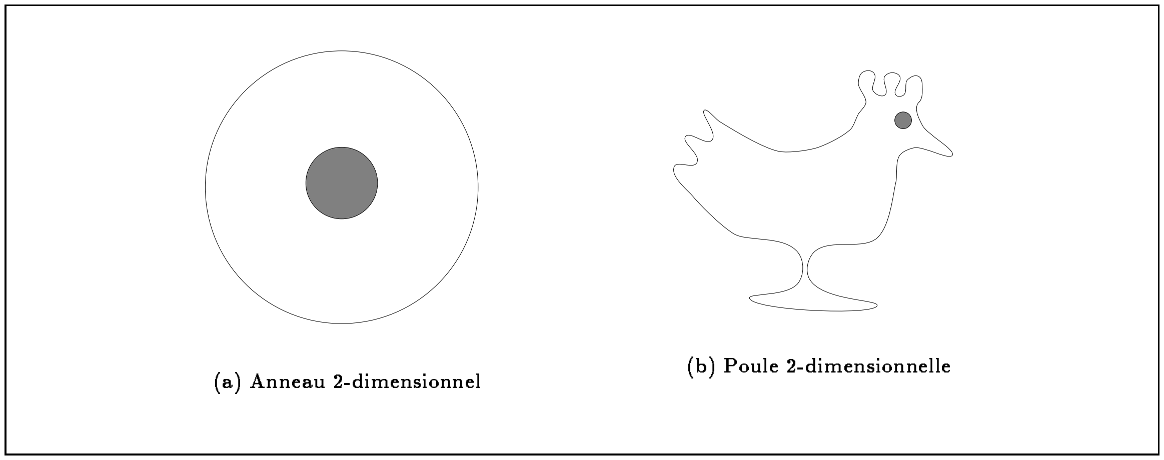


FIG. **2.1** – Déformation continue d'un anneau en une poule 2-dimensionnelle avec un seul trou

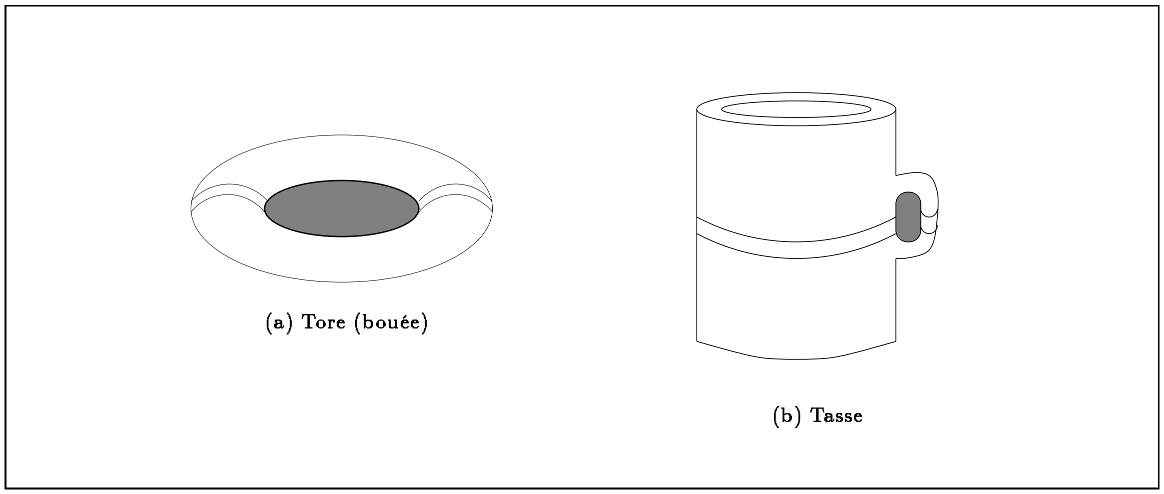


FIG. **2.2** – Déformation continue en 3 dimensions d'un tore (ou bouée) en une tasse avec une seule anse

est fait d'une seule pièce et qu'il possède un trou et deux bords. Un disque, par exemple, possède un seul bord et pas de trou et on peut montrer qu'on ne peut pas déformer continûment un disque en anneau ou vice-versa.

La notion d'homéomorphisme. Un *homéomorphisme* est le terme technique pour désigner ces transformations continues. Les propriétés importantes des espaces topologiques sont celles qui sont invariantes par homéomorphisme. Cependant, le problème de savoir si deux espaces topologiques quelconques sont homéomorphes ou pas est très complexe. En effet, pour montrer que deux espaces sont homéomorphes, il suffit de produire cet homéomorphisme. Par contre, pour montrer que deux espaces ont une nature topologique différente, il faut montrer qu' il n'existe aucun homéomorphisme, ce qui est difficile.

Caractériser les espaces : homotopie et homologie. Pour répondre à ce type de question, les mathématiciens ont développé une approche indirecte : on essaie de caractériser l'espace étudié par une « signature ». Une signature est une *caractéristique invariante par homéomorphisme* de l'espace, comme par exemple la dimension ou bien le nombre de trous dans une surface. Ainsi, si deux espaces ont des signatures différentes, ont sait qu'il ne peuvent pas être homéomorphes. Par contre si les signatures sont les mêmes, on n'est pas assuré de l'homéomorphisme.

Les signatures que nous venons d'évoquer peuvent prendre des formes très variées. Nous en évoquerons deux. Une première approche consiste à distinguer les espaces par les types de chemins différents qu'il contiennent. Deux chemins sont de même type si on peut déformer l'un en l'autre continûment : c'est la notion d'*homotopie*. Les types de chemins correspondent aux éléments d'un groupe (le groupes d'homotopie de l'espace). Or on sait caractériser un groupe (par son cardinal, par ses générateurs, par son index, etc.).

La deuxième approche consiste à classer les espaces en fonction des circuits (des chemins qui bouclent) différents : c'est la théorie de l'*homologie*. L'homologie classe les espaces topologiques en fonction des obstructions (les « trous ») qu'ils contiennent. A partir de cette notion, il est possible de calculer des nombres représentatifs des groupes d'homologie, ce sont les nombres de Betti.

Décrire la structure d'un espace : les espaces cellulaires. Pour pouvoir classer les espaces, encore faut-il avoir pu les décrire ou les construire. Il y a beaucoup façons de construire un espace. Par exemple, ont peut construire un nouvel espace par somme, produit, quotient, recollement, projection (topologique) d'espaces donnés. Une autre façon de faire est de décrire un espace comme « juxtaposition » d'espaces plus élémentaires.

On appelle k-cellule un espace abstrait simple de dimension k et qui ne contient aucun trou. L'idée est que les k-cellules vont constituer les briques de base pour construire un espace. Les k-cellules sont considérées comme des objets formels, avec lesquels on va faire de l'algèbre. Ces objets peuvent avoir plusieurs «réalisations». Par exemple dans $\mathbb{R}^n$, une 0-cellule est un point, une 1-cellule est un segment de droite, une 2-cellule est un triangle, etc. Un cube ou un tétraèdre pleins sont deux exemples de 3-cellule. La réalisation d'une k-cellule est en fait n'importe quel ensemble qui est homéomorphe à $[0, 1]^k$.

Le problème à présent est de combiner ces k-cellules pour obtenir un espace plus compliqué (par exemple un espace qui a des trous). Il y a plusieurs manières d'envisager la combinaison des k-cellules. La *topologie algébrique* introduit la notion de *complexe simplicial* pour formaliser la juxtaposition des k-cellules dans un cadre qui reste purement algébrique et discret (i.e., qui peut servir à l'implémentation d'outil sur un ordinateur).

2.1.2 La naissance de la topologie algébrique

La topologie s'est développée suivant deux branches principales : la topologie dite ensembliste ou analytique, et la *topologie combinatoire* devenue ensuite la *topologie algébrique*. Elles se distinguent par leur nature et leurs méthodes et aussi par leurs origines.

La topologie algébrique en tant que discipline mathématique indépendante, est encore relativement jeune. Un des premiers travaux qui doit être considéré comme marquant le début de la topologie algébrique revient à Euler en 1736 [RS97] qui publia un article sur la solution au problème des ponts de Königsberg (voir figure 2.3). Dans cet article, il était clair qu'Euler considérait ce problème de géométrie comme relevant d'un nouveau type, dans lequel la distance n'était pas pertinente.

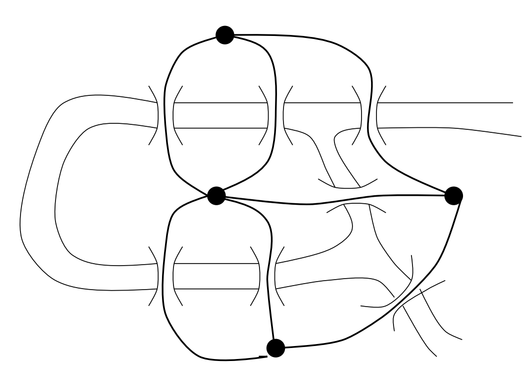


FIG. **2.3** – Les sept ponts de Königsberg. Est-il possible de traverser les 7 ponts de la ville de Königsberg en un seul trajet continu, en ne passant qu'une et une seule fois par chacun d'eux ?

Le pas suivant est également dû à Euler qui établit en 1750 la célèbre formule pour les polyèdres :

$$s - a + f = 2$$

où s désigne le nombre de sommets, a, le nombre d'arêtes, et f, le nombre de faces du polyèdre. Plus tard, en 1813, Lhuilier corrigea la formule d'Euler pour la généraliser aux solides contenant des trous [Lak84] :

$$s - a + f = 2 - 2 \times t$$

où t désigne le nombre de trous du solide. Ce fut le début d'un vaste développement ou plusieurs mathématiciens sur plus d'un siècle, s'occupèrent de la formule d'Euler soit pour en donner une démonstration (qui se révélait toujours incorrecte), soit pour en préciser les conditions de validité (en s'efforçant d'empêcher la construction de contre-exemples considérés comme des « monstres » [Die86]).

C'est dans cette problématique qu'émergea l'idée de topologie algébrique [2] : résoudre par des outils algébriques (et donc souvent discrets) des questions portant sur la structure d'espaces considérés (qui eux sont souvent continus). Avant Euler, il était impossible de penser à des propriétés géométriques sans faire intervenir la notion de *mesure*. Mais dans la formule des polyèdres ci-dessus, seul intervient la configuration des faces, des arêtes et des sommets ; leur forme géométrique exacte (arêtes droites ou courbes) n'intervient pas.

2.2 Quelques éléments de topologie algébrique

La topologie dite algébrique permet d'étudier certaines situations purement topologiques à l'aide de concepts algébriques tels que les groupes. Nous introduisons la notion de complexe simplicial abstrait, puis d'homotopie et de chaîne dans cette structure.

2.2.1 Définition des complexes simpliciaux

La notion de complexe simplicial, permettant de décrire un espace à partir d'espaces plus élémentaires, connaît beaucoup de variations et d'extensions depuis Poincaré [Hen94]. Aussi, nous allons successivement définir la notion de *complexe cellulaire*, qui est un cas particulier d'espace cellulaire, puis celle de *complexe géométrique*, qui est une construction particulièrement intuitive dans $\mathbb{R}^n$ de complexe cellulaire. Nous passerons ensuite à la notion de *complexe simplicial abstrait*, qui généralise la notion de complexe cellulaire. Un complexe géométrique est aussi un exemple de complexe abstrait, ce qui nous permettra de « visualiser » ces entités abstraites.

Complexe cellulaire

Pour construire un espace, nous allons utiliser un ensemble de k-cellules. Cet ensemble C de cellules n'est pas quelconque : il doit satisfaire certaines contraintes :

- le bord d'une cellule est constitué d'une union finie de cellules de C ;

2. C'est B. Listing qui utilisa le premier le mot *topologie*, bien que ses idées sur la topologie étaient principalement dues à Gauss qui choisit de ne rien publier lui-même sur la topologie (Listing était l'élève de Gauss). Poincaré utilisait le terme d'*analysis situ*. D'autres travaux ont contribué à faire émerger la topologie comme branche indépendante, comme par exemple les études menées autour de l'analyse réelle et la formalisation de la notion de limite et de continuité. En 1851, Riemann avait étudié le concept de connectivité de surfaces et introduisit les surfaces de Riemann. Jordan, introduit une autre méthode pour examiner la connectivité d'une surface : compter le nombre de ses circuits. Betti poursuivit ensuite les idées de Riemann, après la mort de celui-ci, en développant la théorie de l'homologie, et introduisit les nombres de Betti. L'idée de connectivité fût réintroduite de manière rigoureuse par Poincaré, qui introduisit ensuite le concept d'*homologie* en donnant une définition un peu plus précise des nombres de Betti (notions dont nous reparlerons en fin de chapitre). Enfin, il introduisit les groupes fondamentaux d'une variété et le concept d'*homotopie* vers 1895.

– l'intersection de deux cellules de C est soit vide soit une cellule de C.

Dans ce cas, on dit que C est un *complexe cellulaire*. Un exemple standard de complexe cellulaire est le *complexe simplicial géométrique*.

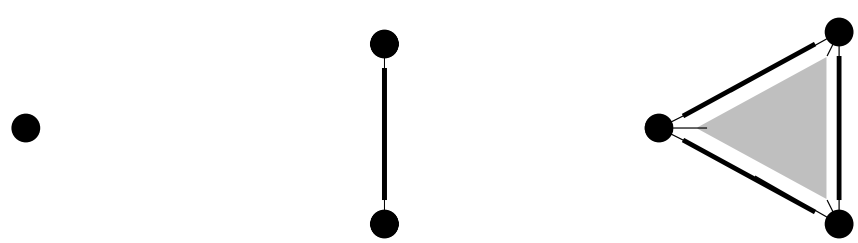
Complexe géométrique

Définition 1 (simplexe géométrique) *Un simplexe géométrique de dimension k dans $\mathbb{R}^n$, avec $k < n$, est une k-cellule constituée de l'enveloppe convexe de $k+1$ points linéairement indépendants. Rappelons que l'enveloppe convexe de $k+1$ points $P_0, ..., P_k$ de $\mathbb{R}^n$ est définie comme l'ensemble* $\{x = \alpha_0 P_0 + ... + \alpha_k P_k,$ avec $0 \leq \alpha_i$ et $\sum_{i=0}^{k} \alpha_i = 1\}$. *Les points P_i sont appelés sommets du simplexe.*

Un *simplexe euclidien* est le simplexe géométrique dont les sommets sont les points $P_i = (\delta_0^i, \delta_1^i, ..., \delta_k^i)$ où $\delta_i^i = 1$ et $\delta_j^i = 0$ si $i \neq j$. On parle de *sommets* et d'*arêtes* pour les simplexes de dimensions 0 et 1. Un simplexe s est une *face* d'un simplexe s' si les sommets de s sont inclus dans les sommets de s'.

Définition 2 (complexe géométrique) *Un complexe géométrique C de $\mathbb{R}^n$ est un ensemble de simplexes géométriques s^p vérifiant : soit $s^p \cap s^q = \emptyset$, soit $s^p \cap s^q = s^r \in C$.*

On peut vérifier aisément qu'un complexe géométrique est un complexe cellulaire, en prenant la définition suivante du bord d'un simplexe : le bord d'un simplexe $(P_0, ..., P_k)$ de dimension $k > 0$ est l'ensemble des (k+1) simplexes de dimension $k-1$: $\{(P_1, ..., P_k), (P_0, P_2, ..., P_k), ..., (P_0, ..., P_{k-1})\}$; et le bord d'un simplexe de dimension 0 est vide. Cette définition correspond bien à l'intuition qu'on a, comme on peut s'en convaincre sur l'exemple des simplexes euclidiens : par exemple, le bord du segment $(0, 1)$ sur $\mathbb{R}$ est bien constitué des deux points 0 et 1.



(a) 0-simplexe

(b) Les deux extrémités du 1-simplexe, sont des 0-simplexes

(c) Le 2-simplexe contient les 0-simplexes et 1-simplexes sur lesquels il s'appuie

FIG. **2.4** – Réalisation géométrique de n-simplexes $n = 0, 1, 2$.

Complexe simplicial abstrait

La notion de complexe cellulaire, ou de complexe géométrique, n'est cependant pas encore assez abstraite : elle fait intervenir la notion de bord, qui est « naturelle » dans le cas d'un complexe géométrique mais qui est plus problématique dans le cas d'un espace abstrait. Un complexe géométrique fait lui intervenir $\mathbb{R}^n$. C'est pourquoi la « bonne » notion, qui nous servira de base dans les implémentations en `Mathematica`, est celle de *complexe simplicial abstrait* ou plus simplement **CS**.

Définition 3 (complexe simplicial abstrait) . *Un complexe simplicial abstrait est un couple (V, C) où V est un ensemble d'éléments (abstraits) appelés* sommets *du complexe, et C un ensemble* *de parties finies de V tels que si $s \in C$, alors toutes les parties $s' \subseteq s$ appartiennent aussi à C. Les éléments de C sont appelés* simplexe abstrait. *La* dimension *d'un simplexe s est le nombre de sommets de s moins un. La dimension du complexe (V, C) est la plus grande dimension des simplexes de C.*

Lorsqu'un p-simplexe s est défini à partir de ses sommets, on le note :

$$s = \langle v^0\, v^1 \ldots v^p \rangle$$

où les v^i sont des éléments de V, l'ensemble des sommets. Si s est un q-simplexe défini à partir d'un sous-ensemble de sommets d'un simplexe s', alors on dit que s est une face de s' et on le note : $s < s'$.

Dans ce qui suit, nous omettrons souvent l'adjectif «abstrait» pour parler des complexes et des simplexes, le contexte permettant de désambigüer. Nous omettrons aussi la plupart du temps la donnée de l'ensemble des sommets d'un complexe. Enfin, nous nous restreignons toujours au cas où l'ensemble des sommets est fini.

Un complexe est donc un ensemble d'ensembles qui est « complet » pour l'inclusion et l'intersection. En effet, si on prend deux simplexes s et s' de C, alors $s \cap s'$ est inclus dans s, et donc appartient aussi par définition à C.

Il est facile d'associer un complexe géométrique à un complexe abstrait $K = (V, C)$. Si n est le cardinal de V, le complexe géométrique est un complexe de $\mathbb{R}^n$. Chaque sommet de V correspond à un point P_i et chaque simplexe de C correspond à un simplexe euclidien. On note $\mid K \mid$ le complexe géométrique ainsi associé à K.

2.2.2 Notions de chemin et d'homotopie

Nous voulons introduire une notion de chemin sur les complexes abstraits, un peu à la manière des chemins de la topologie continue. Un chemin en topologie continue est une fonction continue f de $[0, 1]$ dans l'espace topologique. Le point $f(0)$ correspond au départ du chemin et le point $f(1)$ correspond à l'arrivée. On ne peut pas utiliser cette approche dans le cas des simplexes abstraits car il ne peut pas y avoir d'application continue de $[0, 1]$ dans un ensemble discret autre que les applications constantes. L'idée est qu'un chemin est une suite de sommets reliés par des simplexes.

Une suite finie $\alpha = (P_0, ..., P_n)$ de sommets d'un complexe simplicial K s'appelle chemin polygonal d'origine P_0 et d'extrémité P_n si pour tout couple de points consécutifs (P_i, P_{i+1}), l'ensemble $\{P_i, P_{i+1}\}$ est un simplexe de K. Si $P_0 = P_n$, on dit que α est une chaîne polygonale fermée, elle définit un lacet sur $\mid K \mid$, de point-base P_0. Remarquons que les sommets d'une chaîne polygonale peuvent se retrouver plusieurs fois dans α. Par ailleurs, on peut généraliser au cas des chemins de dimensions supérieures à 1.

Définition 4 (chemin polygonal) *Une suite finie $\alpha = (s_0, ..., s_n)$ de simplexes d'un complexe K s'appelle chemin polygonal d'origine s_0 et d'extrémité s_n si pour tout couple de simplexes consécutifs (s_i, s_{i+1}), $s_i \cap s_{i+1} \neq \emptyset$. La dimension du chemin α est la dimension du plus petit des simplexes s_i plus un.*

Nous voulons à présent définir une notion d'homotopie de chemin polygonal. Nous allons d'abord donner la définition classique dans le cas continu, car celle-ci est peut-être moins connue que les autres notions que nous avons évoquées. Nous définirons ensuite l'homotopie des chemins polygonaux.

Homotopie et groupes d'homotopies dans le cas « continu »

L'outil qui s'intéresse à la comparaison de ces chemins est l'homotopie de chemins. Commençons par définir l'homotopie pour des chemins ayant deux extrémités distinctes.

Définition 5 (homotopie de chemins) *Soient deux chemins l_0 et l_1 de même origine x_0 et de même extrémité x_1. Ils sont dits homotope, s'il existe une application continue $(t, s) \to \Psi(t, s)$ de*

$[0,1] \times [0,1]$ *dans E telle que :*

$$\begin{array}{ll} \forall s, & \Psi(0,s) = x_0,\ \Psi(1,s) = x_1 \\ \forall t, & \Psi(t,0) = l_0,(t)\ \Psi(t,1) = l_1(t) \end{array}$$

L'application Ψ est telle que le montre la figure 2.5

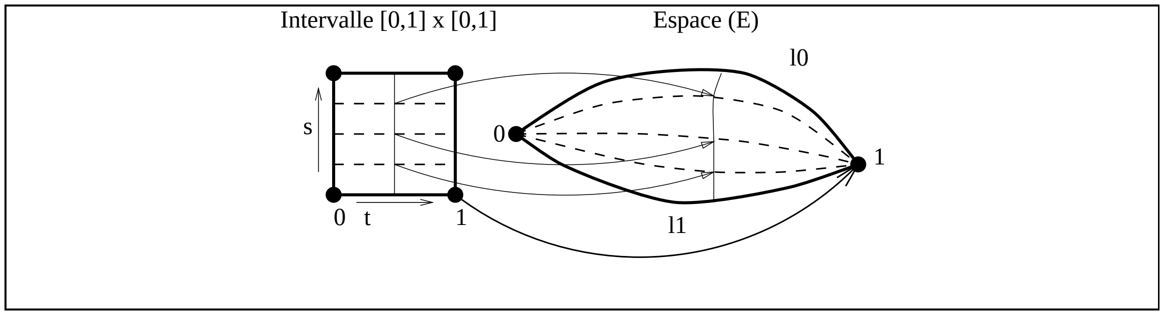


FIG. **2.5** – Définition d'une homotopie entre le chemin l_1 et l_2.

Nous pouvons, de même définir cette homotopie pour des chemins dont l'origine et l'extrémité se réduisent au même point x_0. On obtient alors des lacets du type de ceux de la figure 2.6.

Groupes d'homotopie. Nous pouvons définir des classes de chemins équivalents par déformation. Nous pouvons alors composer ces classes C_1 et C_2 si l'extrémité de la première coïncide avec l'origine de la seconde. On obtiendra alors une classe qui est indépendante des représentants de C_1 et de C_2. Cette loi, appelée $\circ$, est associative.

Si l'on reprend nos chemins fermés sur un point x, on peut également définir des classes de chemins. L'ensemble des classes associées aux lacets de point-base x, muni de la loi de composition $\circ$, constitue un groupe, c'est le groupe d'homotopie de E en x, noté $\Pi(E,x)$.

On peut alors comparer deux groupes d'homotopie $\Pi(E,x)$ et $\Pi(E,y)$. tous les groupes d'homotopie dans un espace connexe par arcs sont isomorphes. Ainsi, à un isomorphisme près, le groupe $\Pi(E,x)$ ne dépend pas du point-base x, on peut parler du groupe $\Pi(E)$, c'est le *groupe fondamental* de E.

Nous pouvons maintenant comparer deux espaces topologiques E et F grâce à cet outil et au théorème suivant :

Théorème 1 *Si f est un homéomorphisme de E sur F, tous deux connexes par arcs, f induit un isomorphisme entre* $\Pi(E)$ *et* $\Pi(F)$.

Autrement dit, le groupe fondamental est un invariant topologique. Attention cependant, si $\Pi(E)$ n'est pas isomorphe à $\Pi(F)$, cela ne nous permet pas de conclure que E et F ne sont pas homéomorphes. (Un exemple d'utilisation de ce théorème est donné par la figure 2.6.)

Groupes d'homotopie n-dimensionnels. Sans entrer dans les détails, nous mentionnons que les résultats précédents concernant les chemins et les homotopies de chemins peuvent être généralisés pour des chemins n-dimensionnels. En effet, nous pouvons généraliser la notion de chemin en considérant les chemins de la figure 2.6 comme 1-dimensionnels. Un chemin n-dimensionnel est alors défini comme $I^n \to E$ où I^n est le pavé unitaire n-dimensionnel de $\Re^n$. Cela permet d'introduire les groupes d'homotopie n-dimensionnels Π_n par analogie avec le groupe fondamental précédent. Le théorème précédent reste vrai pour ces groupes.

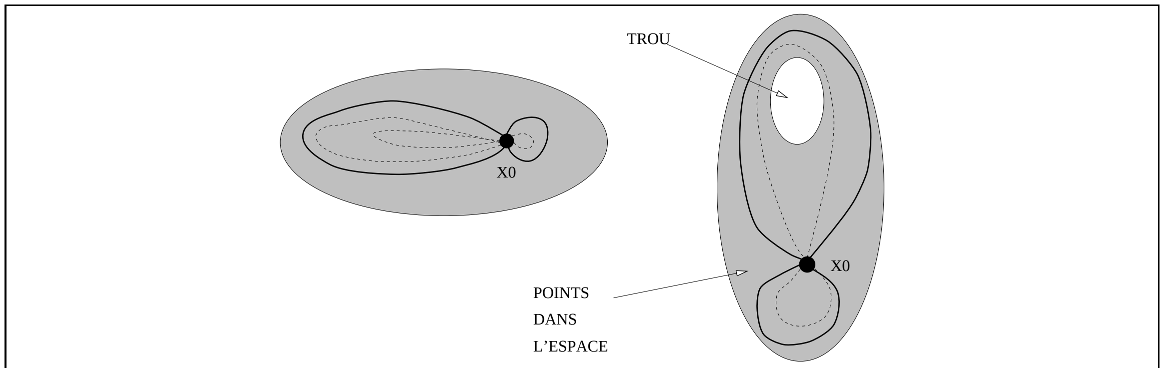


FIG. **2.6** – Deux sous-espaces de $\mathbb{R}^2$ non équivalents topologiquement. La figure illustre deux espaces inclus dans $\mathbb{R}^2$: un disque et un anneau. Intuitivement, on peut se rendre compte que les groupes fondamentaux sont différents. Dans le cas du disque, deux chemins quelconques sont homotopes. Le groupe fondamental se réduit donc au groupe trivial (qui ne comporte qu'un seul élément, l'identité). Dans le cas de l'anneau, les chemins diffèrent fondamentalement par le nombre de tours qu'ils réalisent autour du « trou ». Le groupe fondamental est donc isomorphe à $\mathbb{Z}$. Les deux groupes fondamentaux étant différents, les deux espaces ne sont pas homéomorphes.

Homotopie des chemins polygonaux

On peut développer un analogue des constructions précédentes dans le cas des chemin polygonaux dans les espaces décrits par des complexes abstraits.

Définition 6 (Homotopie de chemins polygonaux) *Deux chemins polygonaux dans un complexe K sont dits homotopes au sens combinatoire si on peut ramener l'un à l'autre en un nombre fini d'opérations, réalisables dans les deux sens, appartenant à l'un des deux types :*

$$\begin{array}{ll} (1) & (..., P, P, ...) \leftrightarrow (..., P, ...) \\ (2) & (..., P, Q, R, ...) \leftrightarrow (..., P, R, ...) \end{array}$$

où P, Q, R sont des simplexes, et où tous les chemins sont des chemins de K.

L'homotopie des chemins polygonaux est une relation d´équivalence entre chemins fermés de même simplexe-base P_0. On peut définir une opération interne sur les classes $\| \alpha \|$ avec :

$$\| \alpha \| \cdot \| \beta \| = \| \alpha \beta \|$$

où $\alpha\beta$ est la chaîne obtenue en prolongeant α avec β : si $\alpha = (P_0, ..., P, ..., P_0)$ et si $\beta = (P_0, ..., Q, ..., P_0)$, alors

$$\alpha \beta = (P_0, ..., P, ..., P_0, P_0, ..., Q, ..., P_0)$$

ce qui nous permet d'introduire un groupe polygonal de K relativement au point $P_0 : \Pi_n(K, P_0)$. On peut montrer que ce groupe est indépendant du point P_0 à un isomorphisme près. Nous retrouvons ainsi, le groupe fondamental $\Pi_n(K)$. De plus, $\Pi_n(\mid K \mid) = \Pi_n(K)$, ce qui veut dire que deux triangulations distinctes d'un même polyèdre ont des groupes isomorphes.

2.2.3 Notion de chaîne et d'homologie

Nous voulons associer un coefficient à des simplexes. La construction mathématique correspondante est celle de chaîne :

Définition 7 (Chaîne) *Soit K un complexe abstrait et G un groupe abélien que l'on notera additivement. Une chaîne n-dimensionnelle sur le simplexe K à coefficient dans G est une fonction*

c_n des n-simplexes de K dans G. L'ensemble des m-chaînes de K à coefficient dans G est noté $C_m(K,G)$. On définit sur cet ensemble une addition :

$$(c_m + c'_m)(s^m) = c_m(s^m) + c'_m(s^m)$$

pour tout m-simplexe s^m (l'addition en partie droite est la loi de groupe de G).

Donnons un exemple. Soit le complexe $K = (\{P,Q,R\}, \langle\langle P,Q\rangle, \langle P,R\rangle, \langle R,Q\rangle, \langle P\rangle, \langle Q\rangle, \langle R\rangle\rangle)$. Et soit G le groupe $(\mathbb{Z},+)$. Alors

$$c_1 = \langle P,Q\rangle \ + \ 2\langle P,Q\rangle$$

et

$$c'_1 = 2\langle P,Q\rangle \ + \ \langle P,Q\rangle$$

sont deux 1-chaînes de K, et

$$c_1 + c'_1 = 3\langle P,Q\rangle \ + \ 3\langle P,Q\rangle$$

est une chaîne de K.

La théorie de l'homologie simpliciale[3] fait intervenir les notions de chaîne et de bord. L'idée est de classer les espaces par le type des cycles qu'il contiennent. Un cycle est une chaîne fermée, et on va distinguer entre les cycles qui sont un bord de ceux qui ne le sont pas.

A chaque p-chaîne c_p on peut associer une $(p-1)$-chaîne ∂c_p appelée le bord de c_p. L'opérateur ∂ est une opération quelconque qui cependant doit vérifier

$$\partial \in C_p \to C_{p-1} \quad \text{et} \quad \partial^2 = 0$$

(un bord n'a pas de bord) et être un homomorphisme de chaîne, c'est-à-dire :

$$\partial(c_p +'_p) = \partial(c_p) + \partial(c'_p) \quad \text{et} \quad \partial(\alpha c_p) = \alpha\, \partial(c_p),\ \alpha \in G$$

On définit alors l'image et le noyau de C_p par :

$$B_{p-1} = \partial(C_p) \quad \text{et} \quad Z_p = \{c_p \in C_p, \partial c_p = 0\}$$

Un élément de Z_p est un p-cycle.

Dans la théorie de l'homologie, on associe à chaque complexe simplicial r-dimensionnel K, pour chacune des valeurs $m \leq r$, le groupe $H^r_m(K) = Z_m/B_m$,. Ce groupe, appelé groupe de Betti m-dimensionnel, est le produit direct de p_m groupes monogènes et d'un certain nombre de groupes cycliques dont les ordres vérifient des propriétés particulières de divisibilité. p_m s'appelle le nombre de Betti m-dimensionnel (voir note 3) Par exemple, pour un espace topologique quelconque le groupe d'homologie 0-dimensionnel reflète le nombre de composantes connexes par arcs.

On peut montrer que les réalisations géométriques de deux complexes simpliciaux abstraits possèdent des groupes de Betti m-dimensionnels isomorphes. Ainsi, on peut parler des groupes de Betti d'un complexe. Ces groupes, comme d'ailleurs les nombres de Betti, sont des invariants topologiques. Leur détermination sur les complexes abstraits a été implémenté en `Mathematica`, bien que nous ne l'utilisions pas dans les applications des deux chapitres suivants. C'est pourquoi nous rejetons la présentation du calcul des nombres de Betti sur les complexes abstraits, et leur interprétation dans notre cadre, en annexe.

3. Cette théorie est présentée plus en détail en annexe.

2.3 Les complexes simpliciaux et les graphes

Les complexes simpliciaux ne sont pas inconnus en informatique : ils sont utilisés par exemple pour la construction de volumes 3D [Elt94] et ont été proposés pour la formalisation des algèbres de processus et la représentation de contraintes spatio-temporelles. Leur introduction dans le domaine du parallélisme a été particulièrement fertile [Pra91, vG91, GJ92, HR95, Gun94].

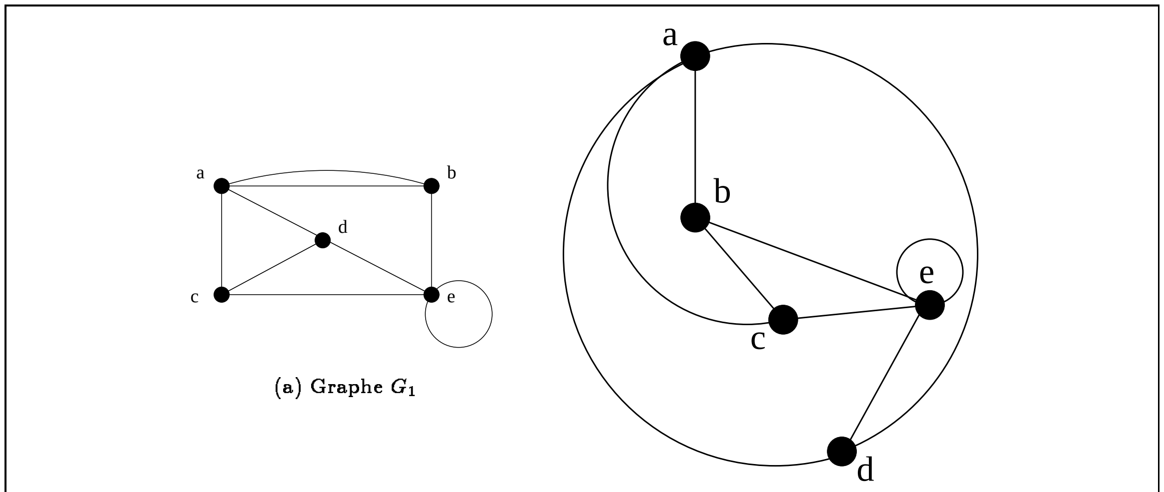
Plus généralement, tous les complexes de dimensions $n < 2$, sont des *graphes*. Par exemple, le problème des ponts de Königsberg est ramené à circuler sur un graphe à quatre nœuds (voir figure 2.3). La théorie actuelle des graphes, les définit seulement à partir de deux ensembles, celui des sommets et celui des arêtes, et d'une relation d'incidence entre sommets et arêtes.

Définition 8 (Graphe orienté) *Le couple $G = (S, A)$, est un* graphe orienté*, dont l'ensemble des sommets est S, et l'ensemble des arêtes est $A \subseteq S \times S$.*

Nous noterons donc que la notion de complexe abstrait généralise la notion de graphe.

On dit que deux sommets sont *adjacents* s'il existe une arête les connectant. Si les sommets d'un graphe sont des points d'un espace topologique, et les arêtes des arcs ou des courbes de Jordan de cet espace, dont les points communs sont des sommets du graphe, on dit que celui-ci est un *graphe topologique*.

On peut définir une bijection entre les ensembles de sommets de deux graphes d'un part, et les ensembles d'arêtes des deux graphes, d'autre part, qui respectent les relations d'incidence. On dit que les graphes sont isomorphes, comme par exemple, les deux graphes de la figure 2.7.



(a) Graphe G_1

(b) Graphe G_2

FIG. **2.7** – L'isomorphisme f entre le graphe G_1 et le graphe G_2 est mis en évidence en nommant de la même manière l'image dans G_2 par f de chacun des sommets de G_1.

2.4 Implémentation des CS en `Mathematica`

Notre but, nous l'avons vu, est d'utiliser les concepts de la topologie algébrique combinatoire que nous venons de présenter, en vue d'une modélisation informatique des processus d'extraction de représentation et de raisonnement diagrammatique.

Nous allons donc développer une bibliothèque d'outils, permettant de manipuler des simplexes et des complexes *colorés*. Un simplexe coloré est une paire constituée d'un simplexe et d'une couleur, une couleur correspondant à un attribut quelconque. De même, un complexe coloré est un complexe composé de simplexes colorés.

Remarquons qu'un complexe coloré n'est pas une chaîne dont les coefficients sont la couleur, car tous les simplexes du complexe n'ont pas forcément la même dimension. Notons, par ailleurs, que l'on retrouve la notion standard de simplexe et de complexe abstrait en ne considérant qu'une seule couleur possible.

Nous définissons ci-dessous :

- des opérations de création de simplexes et de complexes colorés ;
- trois sortes de visualisation graphiques différentes des objets ;
- des observateurs sur ces objets (dimension, sommets, squelettes) ;
- des fonctions permettant le « pattern-matching » d'un complexe dans un autre complexe.

Dans le paragraphe suivant, nous présenterons des exemples d'applications des CS pour la représentation de connaissances.

2.4.1 Le choix du langage `Mathematica`

Nous avons choisi de développer nos applications dans le langage `Mathematica`, pour plusieurs raisons. La première, est que ce langage est très adapté pour le développement rapide de petits exemples, qu'il est facile à manipuler et à présenter.

De plus, `Mathematica` nous permet une visualisation directe des réalisations graphiques des complexes que nous voulons implémenter, avec peu de fonctions graphiques, et dans le même corps de programme.

Enfin, `Mathematica` est un langage pour le calcul symbolique. Ce langage est par nature adapté à l'illustration de notions mathématiques, telles que celles que nous voulons programmer.

Nous avons opté pour une approche fonctionnelle de programmation. En effet, nous souhaitons développer des fonctions correspondant aux structures et au opérations sur ces structures de manière simple et indépendante de toute application pour l'instant.

2.4.2 Création des simplexes et complexes

Nous supposons que le lecteur est familiarisé avec le langage `Mathematica`, ou un autre langage de manipulation symbolique, comme `Maple`, par exemple.

Nous appelons *constructeur*, la tête non interprétée d'un terme. Par convention, le nom des constructeurs est un symbole qui commence par la lettre `c`. Les fonctions qui construisent ces constructeurs débutent par `Creer`, les prédicats se terminent par la lettre `Q`.

Créer des simplexes

La fonction `CreerSimplex` ci-dessous, permet de créer des simplexes à partir de la donnée des sommets. Elle peut associer une couleur, notée `c` à un simplexe donné. Si aucune couleur n'est précisée, le simplexe aura la couleur par défaut `void`. Ce système de couleurs permettra ensuite de distinguer les simplexes d'un complexe qui représentent des relations différentes.

Un simplexe coloré correspond au constructeur `cSimplex`, suivi de la liste triée de ses sommets et de sa couleur.

```
CreerSimplex[cSimplex[l_List, _]] := CreerSimplex[l]
CreerSimplex[l:{_cSimplex..}, c_] := cSimplex[Union@@(#[[1]]&/@l), c]
CreerSimplex[l_List] := CreerSimplex[l, void]

CreerSimplex[l_List, c_] := cSimplex[Union[l], c]
CreerSimplex[cSimplex[l_List, _], c_] := cSimplex[l, c]
CreerSimplex[l__] := CreerSimplex[{l}]
```

Nous aurons également besoin d'une fonction de tri sur les simplexes, nous avons écrit pour cela `SortSimplexFct`.

```
SortSimplexFct[cSimplex[l1_List, _], cSimplex[l2_List, _]] :=
                      Length[l1] > Length[l2]
```

Créer des complexes à partir des simplexes

De la même manière, grâce à la fonction `CreerComplex`, nous construisons des complexes à partir de simplexes. Nous décidons de représenter un complexe non par l'ensemble de ses simplexes, mais uniquement par l'ensemble des simplexes maximaux (maximaux pour la relation d'inclusion). Plus précisément, soit $\overline{K}$, la représentation d'un complexe, alors si $\sigma_1 \in K$, $\sigma_2 \in K$, et $\sigma_1 < \sigma_2$, alors $\overline{\sigma_1} \notin \overline{K}$. $\overline{K}$ est donc l'ensemble des simplexes de K incomparables par $<$. Nous disposons d'une procédure qui, à partir d'un simplexe, fournit tous les sous-simplexes associés. Stocker $\overline{\sigma_1}$ dans la représentation $\overline{K}$ est donc redondant.

Par exemple, supposons que $\mid K \mid$ soit le simplexe euclidien de dimension n, alors $\mid K \mid$ est constitué de 2^n simplexes alors que $\overline{K}$ contient un seul simplexe de n sommets.

```
CreerComplex[c:cComplex[l___], s_cSimplex] :=
         If[DansCQ[s, c], c, Union[cComplex[s, l]]]

CreerComplex[l:{_cSimplex...}] :=
         With[{lc = Union[Color/@l]},
         With[{ll = Sort[Cases[l, cSimplex[_, #]], SortSimplexFct]& /@ lc},
                  Fold [Fold[CreerComplex, #1, #2]&, cComplex[], ll]
                  ]]

CreerComplex[l:_cSimplex..]  := CreerComplex[{l}]
CreerComplex[l:_cComplex..]  := CreerComplex@@Join[l]
```

Exemples d'utilisation

Voici un exemple d'utilisation de ces fonctions de création. Nous pouvons créer un simplexe sans préciser la couleur, sa couleur sera alors `void`. Si nous reprenons le même simplexe, et que nous le redéfinissons avec la couleur `rouge`, la couleur est mise à jour. Nous pouvons encore le redéfinir pour enlever la couleur `rouge`.

```
s1 = CreerSimplex[a, b, c]
CreerSimplex[s1, rouge]
CreerSimplex[s1]

↪ cSimplex[{a,b,c},void]
  cSimplex[{a,b,c},rouge]
  cSimplex[{a,b,c},void]
```

Enfin, les doublons sont enlevés des définitions des simplexes :

```
s2 = CreerSimplex[{a, a, b, c, d, d}, vert]
CreerSimplex[{s1,s2},bleu]
```

```
↪ cSimplex[{a,b,c,d},vert]
  cSimplex[{a,b,c,d},bleu]
```

Pour créer un complexe à partir de simplexes, on peut utiliser pour désigner les simplexes, la notation par leur nom ou bien par leur mode de construction :

```
c1 = CreerComplex[s1, s2, CreerSimplex[a, x], CreerSimplex[{a, x}, rouge],
        CreerSimplex[a, b, c, x, y], CreerSimplex[a, y]]

↪ cComplex[cSimplex[{a,x},rouge],cSimplex[{a,b,c,d},vert],
  cSimplex[{a,b,c,x,y},void]]

c2 =CreerComplex[ CreerSimplex[{a, b, c}, rouge],
        CreerSimplex[{d, e}, vert],
        CreerSimplex[{a, f}, rouge],
        CreerSimplex[{a, c, e}, vert],
        CreerSimplex[{b}, bleu],
        CreerSimplex[{x, y, z, t}, gris],
        CreerSimplex[{c, d, e, f}, bleu] ]

↪ cComplex[cSimplex[{b},bleu], cSimplex[{a,f},rouge], cSimplex[{d,e},vert],
  cSimplex[{a,b,c},rouge], cSimplex[{a,c,e},vert], cSimplex[{c,d,e,f},bleu],
  cSimplex[{t,x,y,z},gris]]
```

Création aléatoire d'un complexe

Enfin, nous nous donnons une dernière fonction capable de créer des complexes au hasard à partir de la donnée d'une dimension et d'un nombre de sommets. Cela sera utile pour tester les fonctions de représentation et de manipulation des complexes. Voici la fonction `RandomSimplex` qui génère des simplexes à partir d'une dimension `dim` et d'un nombre de sommets `p` :

```
RandomSimplex[dim_Integer, p_Integer] :=
        CreerSimplex@@Table[Random[Integer, {1, p}], {1+dim}]
```

Voici la fonction similaire pour construire un complexe au hasard :

```
RandomComplex[dim_Integer, nsimp_Integer, nv_Integer] :=
        CreerComplex@@Table[RandomSimplex[Random[Integer, {0,dim}], nv], {nsimp}]

RandomComplexV[dim_Integer, nsimp_Integer, nv_Integer] /;nv≤26:=
        RandomComplex[dim,nsimp,nv] /.
                    {x_Integer:> FromCharacterCode[ToCharacterCode["A"]+x]}
```

Et voici des exemples d'utilisation de la fonction `RandomComplex`.

```
RandomComplex[3, 5, 10]

↪ cComplex[cSimplex[3,4,void],cSimplex[6,8,void],cSimplex[2,8,9,void],
  cSimplex[3,5,8,void]]

RandomComplex[3, 50, 3]

↪ cComplex[cSimplex[1,2,3,void]]
```

2.4.3 Représentations graphiques

Afin de visualiser les résultats des fonctions précédentes, nous allons implémenter des fonctions qui leur associent des réalisations graphiques.

Nous pouvons représenter une réalisation géométrique des complexes sous la forme de polyèdres connectés. Nous avons déjà introduit la représentation géométrique des simplexes sous cette forme sur la figure 2.4. Les complexes étant composés de simplexes, il est naturel de les représenter de cette façon. Par exemple, le complexe de la figure 2.8 est de dimension 2 et possède deux composantes. On note $\mid K \mid$ la réalisation géométrique du complexe simplicial K.

En général, un complexe de dimension p sera représenté à l'aide d'un polyèdre de dimension p ayant $(p+1)$ arêtes. Cependant, il n'est pas commode de visualiser un complexe de dimension supérieure à 3. Nous verrons comment nous pouvons tout de même en donner une idée à l'aide des trois types de représentations que nous proposons.

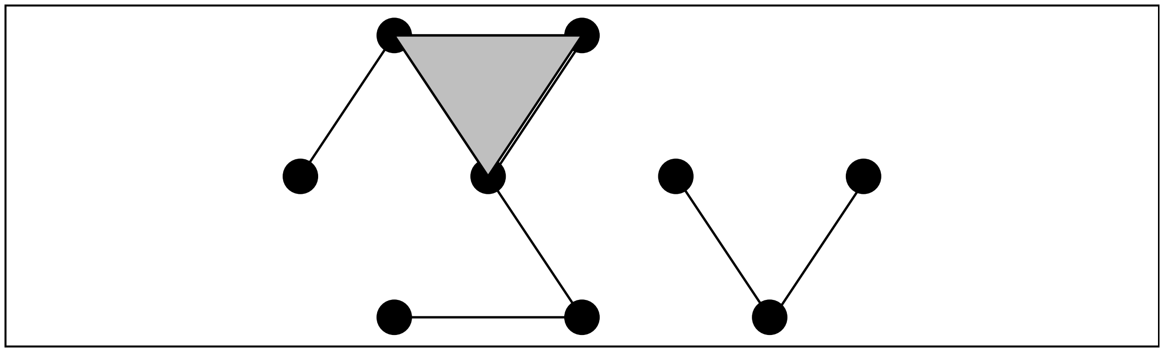


FIG. **2.8** – Complexe simplicial K de dimension 2 avec deux composantes

Visualisation sur un cylindre

Pour cette visualisation, nous positionnons les sommets des simplexes le long d'un cylindre, avec un maximum de trois points par plan de coupe perpendiculaire à l'axe du cylindre. Ainsi, nous aurons au plus un 2-simplexe sur un plan 2-dimensionnel du cylindre.

Il faut procéder pour cela, à un tri préliminaire sur les simplexes, afin de dessiner les plus grands en premier. C'est pourquoi la fonction `ShowComplex`, est définie en faisant intervenir la fonction de tri `SortSimplex`. Nous avons une définition limitée à des complexes contenant des simplexes de dimension inférieure ou égale à trois.

```
ShowComplex[c_cComplex /; Dim[c] ≤3] :=
     With[{som = Sommets[c]},
     With[{renumerote = Table[som[[i+1]] -> i, {i, 0, Length[som] - 1}]},
        {Graphics3D[Sort[List@@c, SortSimplexFct] /.  renumerote] /.  cSimplex ->
GS,
          Graphics3D[Map[Text[StringForm["``", #],
          With[{ls= #/.renumerote}, PtCoord[ls] + Ecart[ls]]]&,
              som]]}
     ]]
ShowComplex[c_cComplex /; Dim[c] >3]:=
  Error["ShowComplex:  not a tridimensional complex:  ", c]
```

La fonction qui trace explicitement la représentation est `GS`, elle est définie à partir des éléments graphiques fournis par `Mathematica`, et par une fonction auxiliaire `PtCoord`.

```
GS[] := {PointSize[0.03]}
GS[{p1_Integer}, c_] :=
   {GraphicColor[c], PointSize[0.03], Point[PtCoord[p1]]}
GS[{p1_Integer, p2_Integer}, c_] :=
    {GraphicColor[c], Thickness[0.01], Line[{PtCoord[p1], PtCoord[p2]}]}
GS[{p1_Integer, p2_Integer, p3_Integer}, c_] :=
    {GraphicColor[c], Polygon[{PtCoord[p1], PtCoord[p2], PtCoord[p3]}]}
GS[{p1_Integer, p2_Integer, p3_Integer, p4_Integer}, c_] :=
    {GraphicColor[c], Polygon[{PtCoord[p1], PtCoord[p2], PtCoord[p3],
           PtCoord[p1], PtCoord[p2], PtCoord[p4],
           PtCoord[p1], PtCoord[p4], PtCoord[p3],
           PtCoord[p4], PtCoord[p2], PtCoord[p3]}]}
```

où la fonction PtCoord est définie simplement comme suit, pour affecter des coordonnées réelles aux points sur le cylindre, ces coordonnées sont définies modulo 3.

```
PtCoord[p_] := Switch[Mod[p, 3],
                      0, {0, 0, 0},
                      1, {1, 0, 0},
                      2, {0, 0, 1}] + {0, Quotient[p, 3], 0}
```

Enfin, les fonctions Ecart et BasicColorRule servant à définir ShowComplex, sont définies par :

```
Ecart[p_] := Switch[Mod[p, 3],
                      0, {-0.1, 0, -0.1},
                      1, {0.1, 0, 0},
                      2, {-0.1, 0, 0.1}]
```

Les couleurs des simplexes, sont traduites en couleurs sur le graphique, ici la règle qui exécute ce changement s'appelle BasicColorRule. Cependant, l'appel de cette fonction passe par la fonction ColorRule, ce qui nous permettra de donner d'autres règles de correspondances en fonction de l'application désirée.

```
BasicColorRule = {void -> GrayLevel[0],
                       vert -> RGBColor[0, 1, 0],
                       rouge -> RGBColor[1, 0, 0],
                       bleu -> RGBColor[0, 0, 1],
                       gris ->  GrayLevel[0.8]};

ColorRule = BasicColorRule;

ListOfColor := ColorRule /.  {Rule[x_ , y_] -> x};
GraphicColor[x_ /; MemberQ[ListOfColor, x]] := x /.  ColorRule
GraphicColor[_] := {}
```

La figure 2.9 donne quelques exemples de complexes représentés sur un cylindre. Le cylindre lui-même n'est pas représenté. Le parallépipède est un cadre généré afin de donner l'idée de perspective. Les complexes qui sont représentés dans la suite, sont générés à l'aide de la fonction RandomComplex.

Il faut être néanmoins prudent avec ce type de représentation car certains simplexes s'interpénètrent alors qu'ils devraient figurer sur des dimensions différentes. Nous sommes, bien entendu limités aux trois dimensions simulées par la perspective sur les deux dimensions de la feuille, alors que généralement, les complexes atteignent des dimensions plus grandes. Le fait de les ramener à une représentation sur un cylindre peut donc créer des ambiguïtés.

Visualisation sur un cercle

Voici une autre fonction de visualisation, que nous avons nommée ViewComplex. Cette visualisation n'est pas limitée à des complexes contenant des simplexes de dimension inférieure à trois. Son inconvénient est de proposer une visualisation un peu moins naturelle que la précédente, mais son avantage est justement de pouvoir représenter les complexes de dimension supérieure à trois.

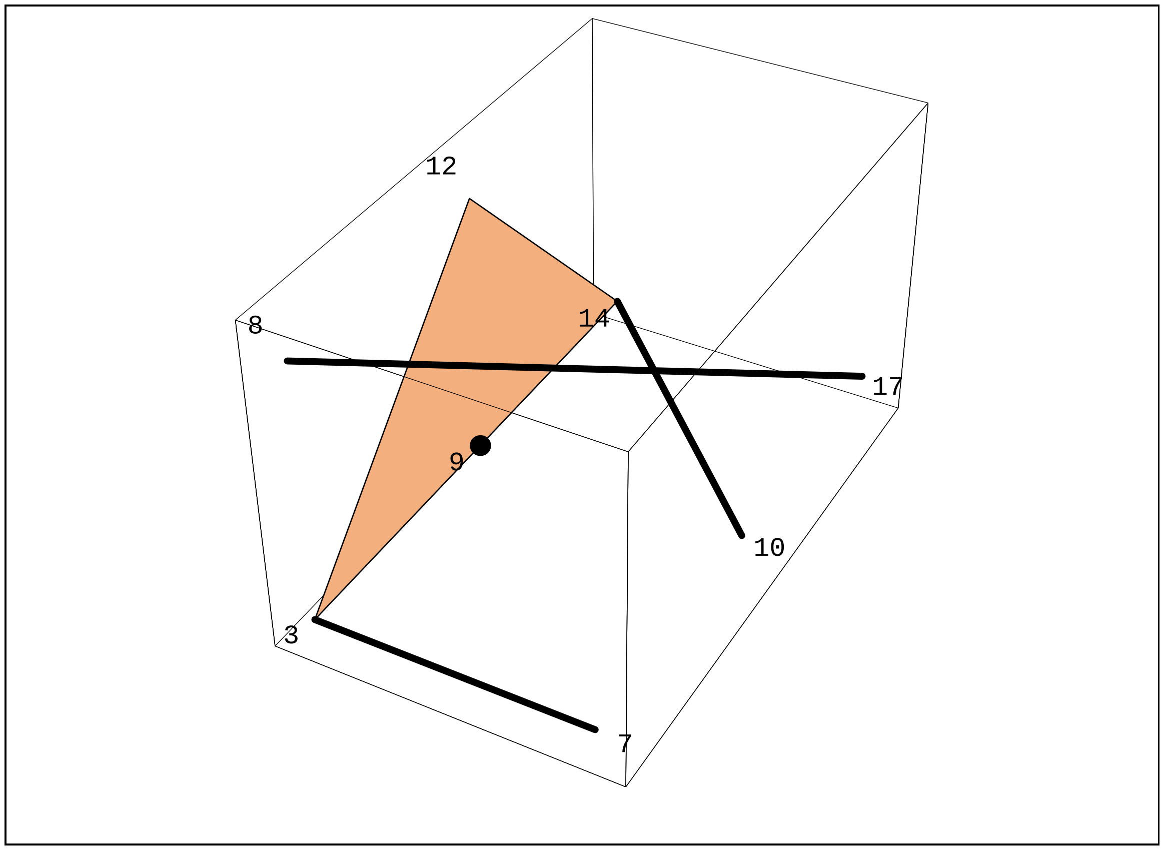


FIG. **2.9** – Représentation du complexe $\langle\langle 3,12,14\rangle,\langle 8,17\rangle,\langle 10,14\rangle,\langle 3,7\rangle,\langle 9\rangle\rangle$, sur un cylindre.

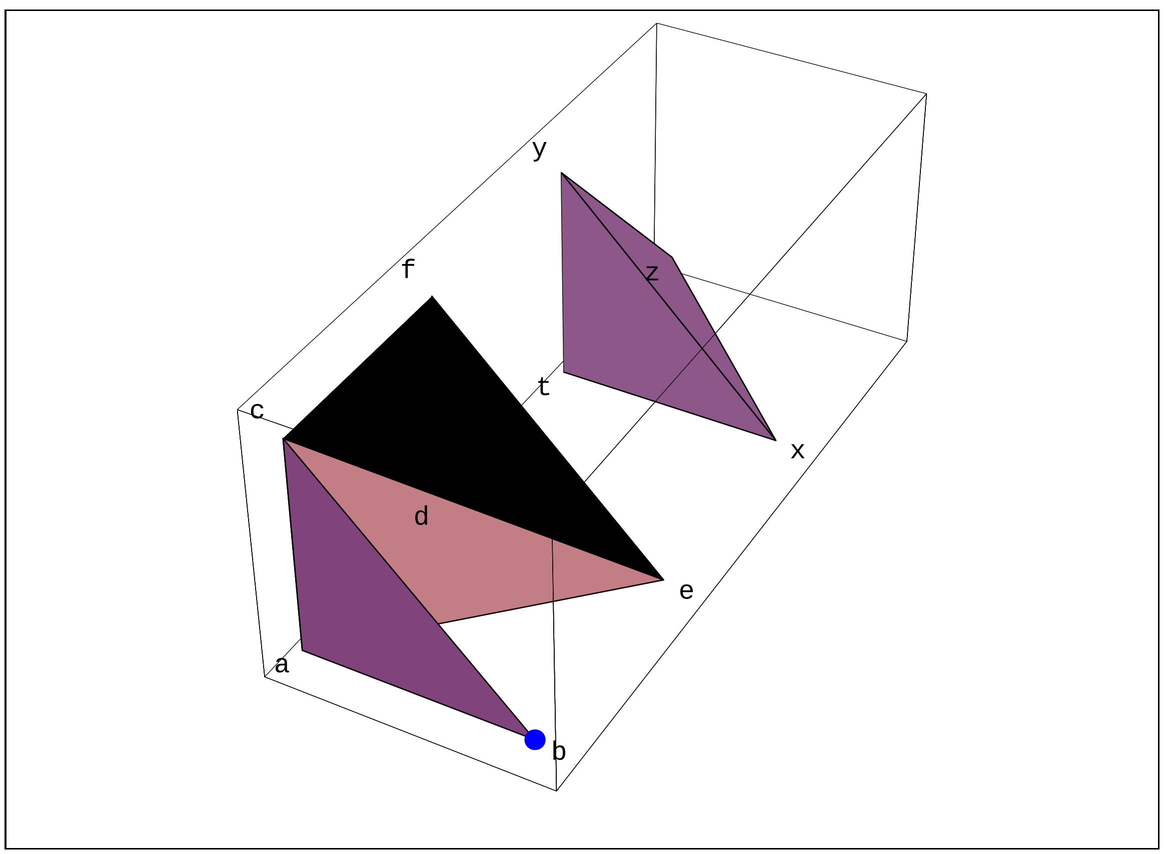


FIG. **2.10** – Représentation du complexe $\langle\langle a,b,c\rangle,\langle a,c,e\rangle,\langle c,d.f,e\rangle,\langle x,y,z,t\rangle,\langle b\rangle\rangle$, sur un cylindre.

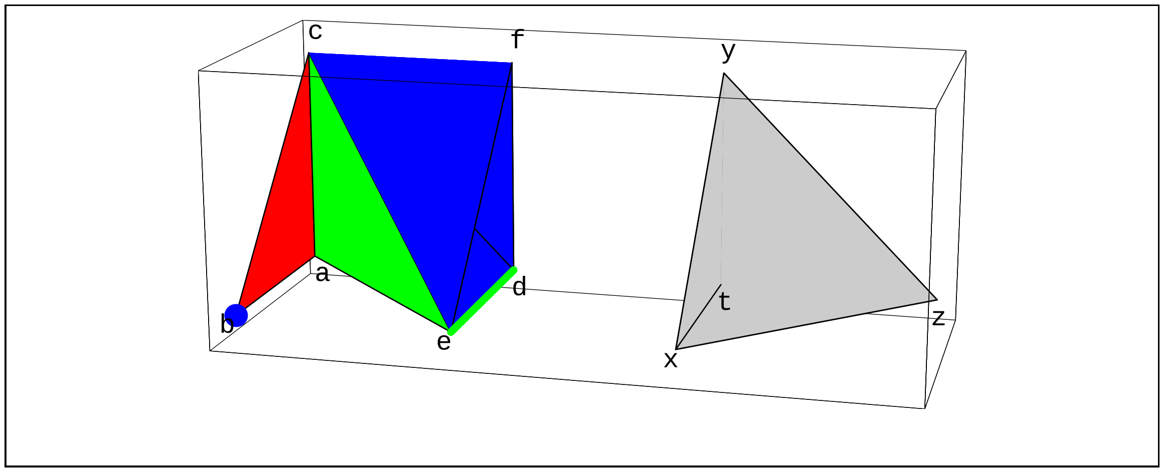


FIG. **2.11** – Représentation du complexe $\langle\langle a,b,c\rangle,\langle a,c,e\rangle,\langle c,d,f,e\rangle,\langle x,y,z,t\rangle,\langle b\rangle\rangle$, avec un autre ordonancement des sommets par rapport à la figure 2.10.

L'idée est simplement de dessiner les sommets du complexe sur un cercle, à distances égales, et de dessiner ensuite le polygone correspondant à la réalisation graphique d'un n-simplexe, c'est-à-dire, un polygone à $(n+1)$ sommets.

Cependant, il faut trouver un procédé qui permette de visualiser tous les simplexes d'un complexe. Pour cela, nous représentons un sommet comme un morceau de droite sur lequel se décale du centre l'arrivée des polyèdres. Ainsi, un polyèdre de grande dimension aura des sommets plus éloignés que ceux de petite dimension. A l'appel de la fonction, on peut indiquer l'ordre des sommets sur le cercle (un ordre par défaut est utilisé sinon).

```
ViewComplex[c_cComplex, som_List] :=
     ViewComplex[List@@c, Join[som, Complement[Sommets[c], som]]]

ViewComplex[c_cComplex] := ViewComplex[List@@c, Sommets[c]]

ViewComplex[l_List, som_List] :=
    With[{ll = Sort[l, SortSimplexFct],d = Max@@(Dim/@l)},
    With[{len = Length[som]},
    With[{renumerote = Table[som[[i+1]] -> i, {i, 0, len-1}]},
          {Graphics[ll /.  renumerote /.  cSimplex -> VS[d, len]],
         Graphics[
          Map[Text[
              StringForm["``",#],(1+0.65 d) CiCoord[len, #/.renumerote]]&,
                      som]]}

ViewSimplexList[l_List] := ViewComplex[l,Sommets[l]]
```

La fonction VS, réalise la véritable tâche de graphisme, à la manière de la fonction GS dans l'implémentation précédente.

```
VS[d_Integer, n_Integer][{p1_Integer}, c_] :=
    {GraphicColor[c],PointSize[0.03], Thickness[0.005],
        Point[CiCoord[n, p1]],Line[{CiCoord[n, p1], (1 + 0.55 d) CiCoord[n, p1]}]}

VS[d_Integer, n_Integer][{p1_Integer, p2_Integer}, c_] :=
    {GraphicColor[c],AbsoluteThickness[2],Line[{CiCoord[n, p1],CiCoord[n, p2]}]}

VS[d_Integer, n_Integer][l_List, c_] :=
    {RandHue[c, (Length[l] - 2)/(d - 1)],
    Polygon[ (1 + 0.45 Length[l]) Map[CiCoord[n, #]&, l] ]}
```

La fonction CiCoord, positionne les points à espacements réguliers sur le cercle :

```
CiCoord[n_Integer, i_Integer] := {N[Cos[i 2 Pi / n]], N[Sin[i 2 Pi / n]]}
RandHue[c_, x_] := If[c === void, Hue[Random[Real, x]], GraphicColor[c]]
```

Voici deux exemples de représentations de complexes sur un cercle. Nous avons choisi de représenter le même complexe que celui que nous avons visualisé à l'aide de la représentation sur le cylindre sur les figures 2.10 et 2.11. Nous avons utilisé deux ordonnancements différents des sommets sur le cercle, le lecteur pourra apprécier la différence, selon ses préférences.

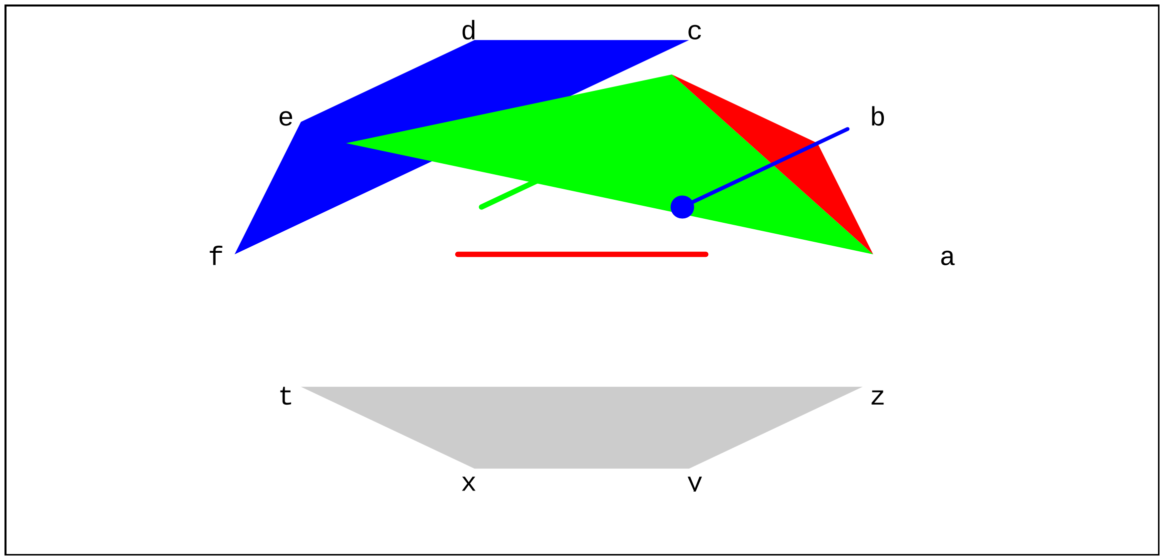


FIG. **2.12** – Représentation du complexe $\langle\langle a,b,c\rangle,\langle a,c,e\rangle,\langle c,d,f,e\rangle,\langle x,y,z,t\rangle,\langle b\rangle\rangle$, sur un cercle

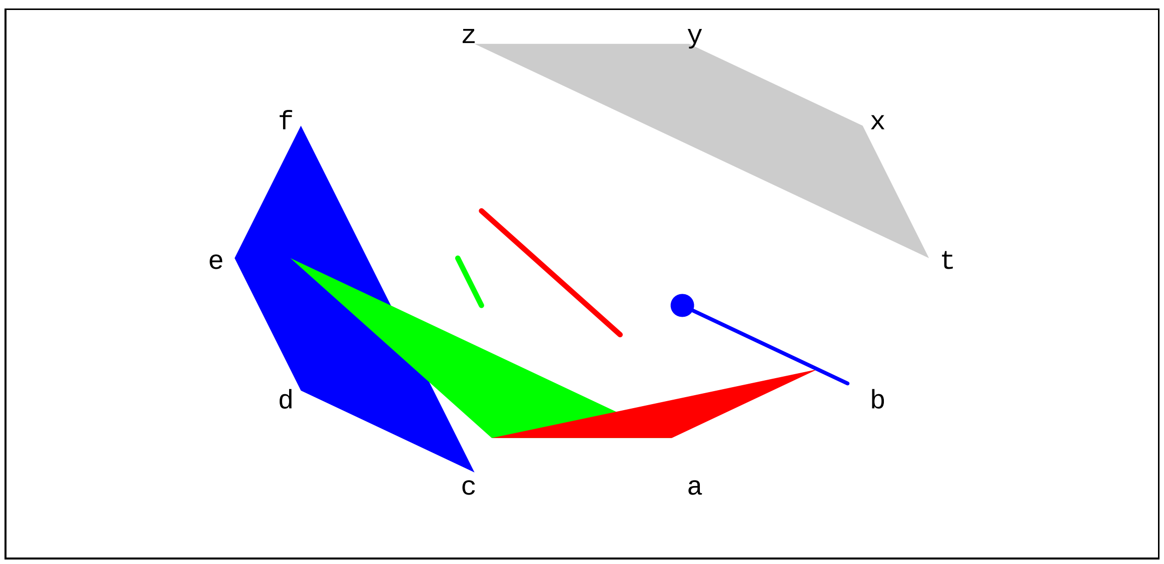


FIG. **2.13** – Représentation du complexe $\langle\langle a,b,c\rangle,\langle a,c,e\rangle,\langle c,d,f,e\rangle,\langle x,y,z,t\rangle,\langle b\rangle\rangle$, sur un cercle, vu d'un deuxième angle par rapport à la figure 2.10

Visualisation à l'aide d'un diagramme de Hasse

Il est possible de représenter une relation d'ordre partiel sur les éléments d'un ensemble, par un graphe prenant pour sommets, les éléments de l'ensemble. Cependant, ce graphe peut être vite illisible, du fait du grand nombre d'arêtes.

Une solution pour améliorer la lisibilité du graphe, consiste alors à ne pas mentionner les arêtes redondantes. En effet, une relation d'ordre partiel vérifie la relation : $a \in E, b \in E, c \in E$, alors $a < b, b < c \Rightarrow a < c$. Il est donc inutile de faire figurer les trois arcs (a,c), (a,b) et (b,c), sur le graphe puisque l'arc (a, c) peut être déduit des autres. Un graphe sur lequel les arcs (a, c) sont économisés, s'appelle un *diagramme de Hasse*, ou diagramme transitif.

Les complexes possèdent cette propriété de transitivité, puisqu'un complexe qui est contenu dans un deuxième, lui-même contenu dans un troisième, est forcément contenu dans ce troisième. En fait, on peut associer de manière canonique un treillis à un complexe : les éléments du treillis sont les simplexes ordonnés par inclusion. Nous allons maintenant exploiter cette propriété et dessiner un complexe sous forme d'un diagramme de Hasse. Nous utilisons un paquetage `Mathematica` pour dessiner ce diagramme : il s'agit de `Combinatorica`.

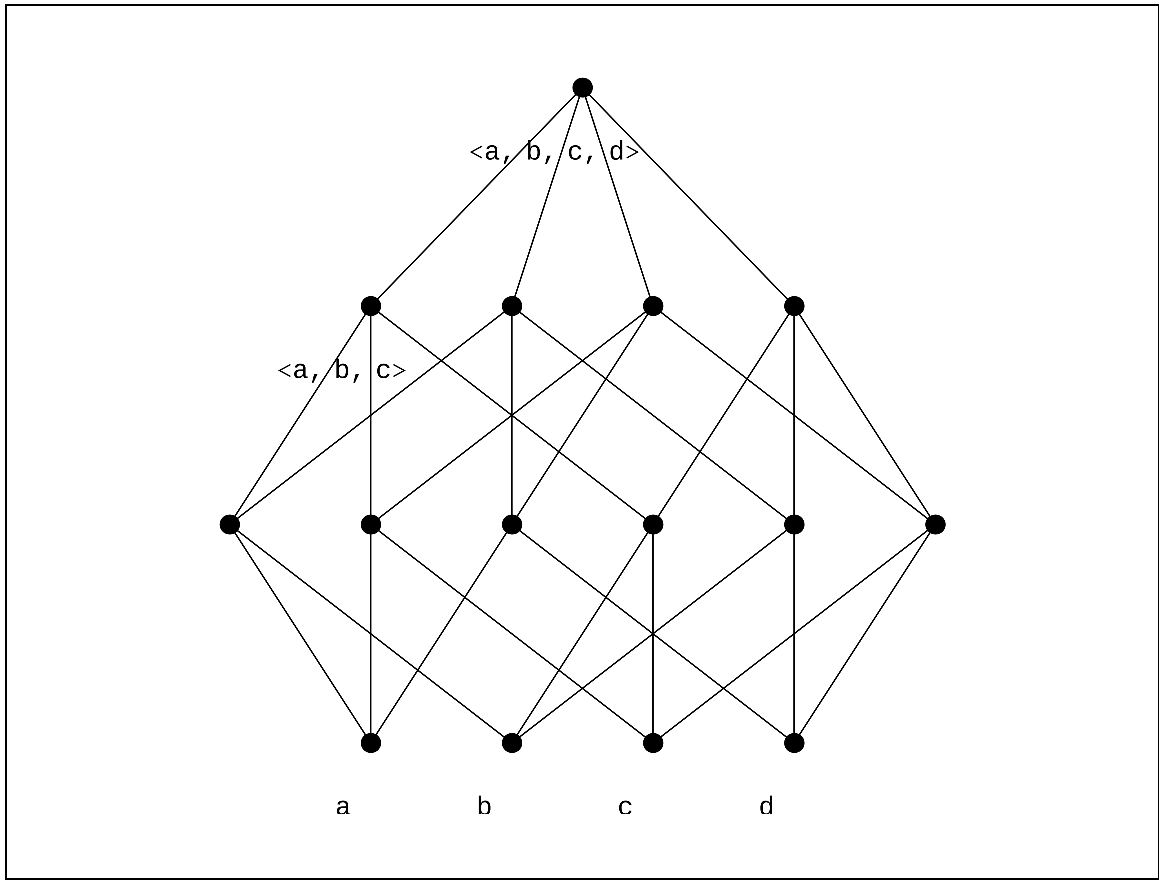


FIG. **2.14** — Visualisation sous la forme d'un diagramme de Hasse : le complexe représenté est $\langle(\langle a, b, c, d\rangle, rouge), (\langle a, b, c\rangle, vert)\rangle$. Chaque point du diagramme correspond à un simplexe. Les simplexes sont présentés par étage, chaque étage contenant les simplexes de même dimension. Les simplexes de dimension 0 sont au rez-de-chaussée. Un simplexe s de l'étage $(n+1)$ est relié à un simplexe s' d'un étage inférieur si s' est une face de s, indépendamment de la couleur. Les noms des sommets sont indiqués au rez-de-chaussée puisque ce sont des 0-simplexes. Le nom des simplexes maximaux est aussi indiqué entre $<>$. Attention, nous tenons compte de la couleur pour attribuer les noms, ainsi même si $\langle a, b, c\rangle \subset \langle a, b, c, d\rangle$, ils sont tous les deux étiquettés sur la figure, car ils ont des couleurs différentes.

Les points du diagramme seront l'ensemble des parties du complexe, l'ensemble des simplexes qu'il contient. La fonction `Mathematica`, `HasseDiagram`, existe déjà dans le package `Combinatorica` produit un diagramme de Hasse à partir d'un treillis.

```
DisplayComplex[c_]:=
    With[{ls=Sommets/@(List@@c)},
        With[{somTreillis=Union[Flatten[PartieDe/@ls,1]]},
            With[{label=Nom[c]/@somTreillis},
                ShowLabeledGraph[
                    HasseDiagram[MakeGraph[somTreillis,
                        ((Intersection[#2,#1]===#1)&&(#1 != #2))&]],
                    label ]
        ]]]
Nom[c_cComplex][l_List]:=
    With[{nm=Cases[c,cSimplex[l,_]:>l]},
        If[nm==={},
            If[Length[l]===1,First[l],""],Plus@@nm]]
```

La figure 2.14 donne un exemple de visualisation, à l'aide d'un diagramme de Hasse, du complexe

$$\langle\langle a,b,c,d\rangle,\langle a,b,c\rangle,\langle a\rangle,\langle b\rangle,\langle c\rangle,\langle d\rangle\rangle$$

2.4.4 Observateurs sur les complexes colorés

Un observateur est une fonction permettant d'extraire un attribut d'une structure de données. Les observateurs intéressants concernant les complexes, que nous avons implémentés, sont les suivants :

- sa couleur ;
- ses sommets ;
- sa dimension ;
- son squelette, i.e. ses simplexes de dimension donnée.

La couleur d'un complexe

La couleur est un attribut très facile à récupérer, à cause de la structure de définition des simplexes et complexes, voici la fonction `Color`, qui l'extrait :

```
Color[cSimplex[_, c_]] := c
```

Les sommets d'un complexe

Les sommets d'un simplexe sont donnés par l'ensemble des sommets qui apparaissent dans la structure `cSimplex`, la fonction `Sommets`, les extrait :

```
Sommets[cSimplex[l_, _]] := l
Sommets[c_cComplex] := Union@@(Sommets /@ c)
Sommets[l_List] := Union@@(Sommets /@ l)
```

La fonction `SommetsC`, réalise la même tâche, mais en prenant précisant la couleur des sommets. Cela revient à chercher les sommets des simplexes impliqués dans une relation donnée.

```
SommetsC[cSimplex[l_,col_]] := Union[{#,col}&/@l]
SommetsC[c_cComplex] := Union@@(SommetsC /@ c)
```

Voici des exemples d'utilisation des fonctions avec et sans précision de la couleur :

```
Sommets[c1]
SommetsC[c1]
```

```
↪ {a,b,c,d,x,y}
  {{a,rouge},{a,vert},{a,void},{b,vert},{b,void},{c,vert},
  {c,void},{d,vert},{x, rouge},{x,void},{y,void}}
```

Fonctions d'inclusion des simplexes et complexes

Les fonctions DansSQ et DansCQ, testent si un simplexe est contenu dans un autre simplexe, en testant seulement les sommets pour DansSQ, et en testant en plus la couleur pour DansCQ :

```
DansSQ[cSimplex[l1_, _], cSimplex[l2_, _]] :=
                (Length[l1]≤Length[l2])&&(Complement[l1, l2] === {})
DansCQ[s1_cSimplex, s2_cSimplex] :=
   (Color[s1] === Color[s2]) && DansSQ[s1, s2]

DansSQ[s_cSimplex, c_cComplex] := LazyOr[DansSQ[s, #]&, c]
DansCQ[s_cSimplex, c_cComplex] := LazyOr[DansCQ[s, #]&, c]

DansSQ[c1_cComplex, c2_cComplex] := LazyAnd[DansSQ[#, c2]&, c1]
DansCQ[c1_cComplex, c2_cComplex] := LazyAnd[DansCQ[#, c2]&, c1]
```

Voici des exemples d'utilisation. s1 est bien contenu dans s2, mais pas l'inverse. Il n'est plus contenu dans s2 lorsque l'on impose la contrainte de la couleur.

```
DansSQ[s1, s2]
DansSQ[s2, s1]
DansCQ[s1, s2]

↪ True
  False
  False
```

Dimension des simplexes et des complexes

La fonction Dim, donne la dimension d'un simplexe ou d'un complexe (dimension du plus grand simplexe du complexe).

```
Dim[cSimplex[l_, _]] := Length[l] - 1
Dim[cComplex[]] := -1
Dim[c_cComplex] := Max@@(Dim/@c)
```

Voici un exemple pour le complexe c1 :

```
Dim[c1]

↪ 4
```

Squelette d'un complexe

Enfin, nous définissons une fonction capable d'extraire le squelette d'un complexe, pour une dimension donnée. Le squelette pour une dimension est la liste des simplexes appartenant au complexe et ayant la dimension voulue. La fonction Squelette réalise cela :

```
Squelette[i_Integer][cSimplex[l_List, c_]] := Map[CreerSimplex[#, c]&,
SousEnsembles[i+1][l]];
Squelette[i_Integer][x_cComplex] := CreerComplex[ Flatten[Squelette[i]/@(List@@x)]];
```

Dans l'exemple suivant, nous construisons un complexe au hasard à l'aide de la fonction RandomComplex, puis nous demandons le squelette de ce complexe pour les dimensions de 0 à 3 :

```
c3 = RandomComplex[2, 5, 10]
Squelette[0][c3]
Squelette[1][c3]
Squelette[2][c3]
Squelette[3][c3]
```

Voici les sorties successives obtenues pour ces commandes :

```
↪ cComplex[cSimplex[{8},void],cSimplex[{4,9},void],cSimplex[{7,9},void],
  cSimplex[{1,6,9},void],cSimplex[{2,4,6},void]]
```

Le complexe au hasard est créé, avec la couleur `void`, par défaut.

```
↪ cComplex[cSimplex[{1},void],cSimplex[{2},void],cSimplex[{4},void],
  cSimplex[{6},void],cSimplex[{7},void],cSimplex[{8},void],cSimplex[{9},void]]
```

À la dimension 0, le squelette est la liste des sommets du complexe, avec la couleur `void`.

```
↪ cComplex[cSimplex[{1,6},void],cSimplex[{1,9},void],cSimplex[{2,4},void],
  cSimplex[{2,6},void],cSimplex[{4,6},void],cSimplex[{4,9},void],
  cSimplex[{6,9},void],cSimplex[{7,9},void]]
```

À la dimension 1, ce sont tous les 1-simplexes du complexe.

```
↪ cComplex[cSimplex[{1,6,9},void],cSimplex[{2,4,6},void]]
```

À la dimension 2, ce sont tous les 2-simplexes du complexe.

```
↪ cComplex[]
```

Il n'y a aucun simplexe de dimension 3 dans le complexe puisque nous l'avons construit en lui demandant d'avoir la dimension 2.

2.4.5 Comparaison de deux complexes par appariement

`FindMatch` cherche les sous-complexes de `c` qui sont appariables au complexe `p` dans lequel on a le droit de substituer les sommets de la liste `var`. La fonction `FindMatchS`, réalise `FindMatch` seulement sur les sommets, tandis que `FindMatchC`, réalise `FindMatch`, en vérifiant que les couleurs des complexes coïncident aussi. Cette fonction permet de chercher dans un complexe, une configuration décrite par un pattern où les variables sont des sommets.

```
FindMatchS[c_cComplex, p_cComplex, var_List]:=FindMatch[c,p,var,DansSQ]
FindMatchC[c_cComplex, p_cComplex, var_List]:=FindMatch[c,p,var,DansCQ]

FindMatch[c_cComplex, p_cComplex, var_List,test_] :=
    With[{sc = Sommets[c],sp=Sommets[p]},
    With[{fixe=Complement[sp,var]},
    With[{candidat=Complement[sc,fixe]},
    With[{subs=Appariement[var,candidat]},
                    Union[ Select[Substitution[#,p]&/@subs,test[#,c]&] ]
    ]]]]
```

où `Appariement` est définie par :

```
Appariement[var_List, candidat_List]:=
    If[Length[var]>Length[candidat],{},Appariement[var,candidat,{{}}]]
Appariement[{},candidat_List,rep_List]:=rep
Appariement[{x_,l___},candidat_List,rep_List]:=
    With[{subs=Appariement2[x,candidat]},
        Join@@Map[Appariement[{l},#[[2]],Map[AddSubs[#[[1]]],rep]]&,subs]]
AddSubs[x_][l_List]:=Join[{x},l]
Appariement2[x_,l_List]:=MapIndexed[{x->#1,Delete[l,#2[[1]]]}&,l]
Substitution[s_List,c_cComplex]:=
    (c/.s)/.{cComplex->CreerComplex,cSimplex->CreerSimplex}
```

`Appariements` cherche les sommets différents entre `p` et `var`, ils seront `fixe` et ne pourront pas être remplacés. Les sommets de `c` qui ne sont pas dans les sommets fixes,seront `candidat` pour le remplacement.

Une fois les appariements déterminés, `Substitution` créé le complexe apparié grâce aux substitutions. Voici des exemples :

```
FindMatchS[c1,CreerComplex[CreerSimplex[a,b,w]],{w}]

↪ {cComplex[cSimplex[{a,b,c},void]],cComplex[cSimplex[{a,b,d},void]],
   cComplex[cSimplex[{a,b,x},void]],cComplex[cSimplex[{a,b,y},void]]}
```

Dans l'exemple ci-dessus, nous cherchons dans `c1`, les 3-simplexes contenant les sommets `a` et `b`. Le résultat est donné sous la forme d'un complexe, il y a quatre complexes solutions. Voici d'autres exemples :

```
FindMatchS[c1,CreerComplex[CreerSimplex[{a,b,w},vert]],{w}]

↪ {cComplex[cSimplex[{a,b,c},vert]],cComplex[cSimplex[{a,b,d},vert]],
   cComplex[cSimplex[{a,b,x},vert]],cComplex[cSimplex[{a,b,y},vert]]}

FindMatchC[c1, CreerComplex[CreerSimplex[{a,v},vert], CreerSimplex[{a,w},rouge]],{v,w}]

↪ {cComplex[cSimplex[{a,b},vert],cSimplex[{a,x},rouge]],
   cComplex[cSimplex[{a,c},vert],cSimplex[{a,x},rouge]],
   cComplex[cSimplex[{a,d},vert],cSimplex[{a,x},rouge]]}
```

2.5 La représentation à l'aide de Complexes Simpliciaux : la Q-Analyse

Après avoir introduit quelques concepts élémentaires de topologie algébrique, nous allons à présent montrer comment il est possible de représenter des relations à l'aide des complexes simpliciaux. **Montrons ainsi comment on peut associer une représentation diagrammatique à une relation binaire et à un ensemble de prédicats.** Ces modes de représentation seront utilisés dans les applications des deux chapitres suivant.

Au début des années 70, R. Atkin a proposé de représenter une relation binaire R entre deux ensembles quelconques par un complexe simplicial : c'est la Q-analyse [Atk74, Atk81, Joh91a]. Les travaux autour de la Q-analyse ont surtout porté sur les domaines de la modélisation des relations sociales [Atk74, Gou80, Cas82], l'analyse des interactions entre agents [Dor86, PLC91, Cas92, chap. 8], l'analyse de trafic [Joh91b], la reconnaissance des formes [Joh90] et l'analyse des positions aux échecs [Atk76].

Le peu d'impact de cette approche dans le domaine de l'IA classique tient à deux difficultés. La première est que, bien que les concepts nécessaires soient élémentaires en topologie algébrique, cette branche des mathématiques est récente et ne fait pas partie du cursus habituel de l'informaticien. C'est principalement des mathématiciens qui ont développé les applications de la Q-analyse et il se sont surtout cantonnés à la représentation de problèmes en sciences humaines et sociales, sans aborder la manipulation algorithmique de ces représentations. L'aspect « raisonnement automatique » a donc largement été ignoré.

La deuxième difficulté est que les algorithmes naïfs sur ces objets, comme par exemple le calcul des nombres de Betti associé à un CS, sont des algorithmes combinatoires coûteux. La puissance de calcul nécessaire n'est disponible que depuis récemment (grâce à l'aide des machines parallèles par exemple). Et les algorithmes « non-naïfs » qui sont efficaces ne sont étudiés que depuis peu (voir par exemple [DE93] pour le calcul rapide des nombres de Betti en dimension 3 et [DS96] pour l'analyse structurelle des complexes simpliciaux dans $\mathbb{R}^3$).

2.5.1 Relations binaires

Une *relation* binaire d'un ensemble A vers un ensemble B est une règle λ qui associe des éléments de B à des éléments de A. Pour déterminer complètement λ, il suffit de connaître tous les paires d'éléments (a,b) où $a \in A, b \in B$ qui vérifient le prédicat défini par la règle λ. Ainsi, λ est entièrement définie par un sous-ensemble de $A \times B$. La relation associée à λ et partant de B pour retourner dans A est appelée son *inverse* et est notée λ^{-1}.

Une relation peut être représentée à l'aide d'un tableau prenant des valeurs binaires, sa *matrice d'incidence* Λ. Prenons un exemple : soit $A = \{a_1, a_2, a_3\}$ et $B = \{b_1, b_2, b_3\}$. On peut définir :

$$\lambda \subset (A \times B), \quad \lambda = \{(a_1, b_1), (a_2, b_3), (a_3, b_3)\}$$

Alors $\Lambda = \lambda_{ij}$ sera une matrice $Card(A) \times Card(B)$, donnée par :

λ	b_1	b_2	b_3
a_1	1	0	0
a_2	0	0	1
a_3	0	0	1

où $\Lambda_{ij} = 1$ si l'élément $(a_i, b_j) \in \lambda$ et 0 sinon.

Une fonction f, est un cas particulier de relation pour laquelle il correspond au maximum un élément dans B à chaque élément de A. Le domaine de f est la restriction de A aux éléments $a_i \in A$ pour lesquels $f(a_i) \in B$. Lorsque l'on a la propriété :

$$f(a_i) = f(a_j) \Rightarrow a_i = a_j$$

alors f est une *injection*. Lorsque, tous les éléments de B sont une image d'un élément de A par f, on parle de *surjection*. Lorsque l'on a à la fois une injection et une surjection, on dit que f est une bijection, ou encore que f est *bijective*.

Les autres propriétés intéressantes que peut présenter une relation sont :

Réflexive	$(a_i, a_i) \in \lambda, \forall a_i$
Symétrique	$(a_i, a_j) \in \lambda \Rightarrow (a_j, a_i) \in \lambda$
Transitive	$(a_i, a_j) \in \lambda et (a_j, a_k) \in \lambda \Rightarrow (a_i, a_k) \in \lambda$

Si une relation f est transitive, réflexive, symétrique alors elle est appelée *relation d'équivalence*, elle a pour effet de partitionner A. Une partition est une division de A en sous-ensembles d'intersections nulles et dont l'union redonne exactement A. Les sous-ensembles de A créés par partition sont les classes d'équivalence de λ sur A.

2.5.2 Représentation des relations binaires par un CS

La matrice Λ_{ij} que nous avons associée à λ ne peut correspondre qu'à une relation binaire. Nous allons nous intéresser à ces relations binaires, spécifiées sous forme de matrice, et voir quel est leur lien avec les complexes simpliciaux.

Supposons que Λ contienne m lignes et n colones, c'est à dire que $Card(A) = m, Card(B) = n$. Fixons B comme l'ensemble des sommets, ainsi un ensemble de $(p+1)$ éléments de B formera un p-simplexe. Prenons maintenant les éléments de A qui sont impliqués dans la relation λ. Soit a_i l'un d'entre eux. Il est alors possible qu'il apparaisse à plusieurs endroits dans l'ensemble de définition de λ, par exemple :

$$\lambda = \{..., (a_i, b_j), ..., (a_i, b_k), ..., (a_i, b_l), ...\}$$

On peut dire que a_i est $\lambda-$relié à b_j, b_k, b_l. Ce fait sera visible sur la matrice d'incidence comme l'ensemble de tous les 1 de la ligne correspondant à a_i. L'ensemble b_j, b_k, b_l des éléments liés à a_i est inclus dans B, ce sont donc des sommets. Ainsi, on peut définir un simplexe associé à a_i et qui prend appui sur trois sommets, b_j, b_k, b_l. Chaque ligne de la matrice définit un simplexe, l'ensemble de ces simplexes étant caractéristique de λ, on dit alors qu'ils forment le complexe simplicial associé à λ. En prenant par exemple pour λ :

$$\lambda = \{(a_1, b_1), (a_1, b_3), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_2), (a_3, b_3)\}$$

alors nous obtenons la matrice d'incidence Λ associée à λ :

λ	b_1	b_2	b_3
a_1	1	0	1
a_2	0	1	1
a_3	1	1	1

On peut représenter a_3 par un 2-simplexe défini par les trois sommets :

$$a_3 = \langle b_1, b_2, b_3 \rangle$$

puisque a_3 est en λ-relation avec ces trois sommets. De même, nous représentons a_1 et a_2 par deux 1-simplexes :

$$a_1 = \langle b_1, b_3 \rangle \quad a_2 = \langle b_2, b_3 \rangle$$

Si nous les représentons sous forme de polyèdres, le complexe simplicial associé à λ est alors celui de la figure 2.15. Nous avons représenté les sommets b_1, b_2 et b_3, et les arcs qui les relient. La face grisée représente le 2-simplexe associé à a_3, les deux segments en trait fins représentent les 1-simplexes associés à a_2 et a_1. En fait, ces segments sont exactement les arêtes $\langle b_1, b_3 \rangle$ et $\langle b_2, b_3 \rangle$, mais nous les avons écartés pour la clarté du schéma, de même pour la face associée à a_3.

Ainsi, une relation λ de A vers B, avec $Card(A) < \infty$, $Card(B) < \infty$ et $\lambda \subset (A \times B)$ définit un complexe simplicial noté $K_A(B, \lambda)$. De la même façon, λ^{-1}, la relation inverse associée à λ, définit un complexe simplicial noté $K_B(A, \lambda^{-1})$. On dira alors que $K_A(B, \lambda)$ et $K_B(A, \lambda^{-1})$ sont conjugués.

2.5.3 Un exemple complet : le complexe simplicial abstrait des fleurs et des couleurs

Prenons un exemple simple de relation λ pour illustrer notre propos. Considérons l'ensemble F, de types de fleurs [Cas94] :

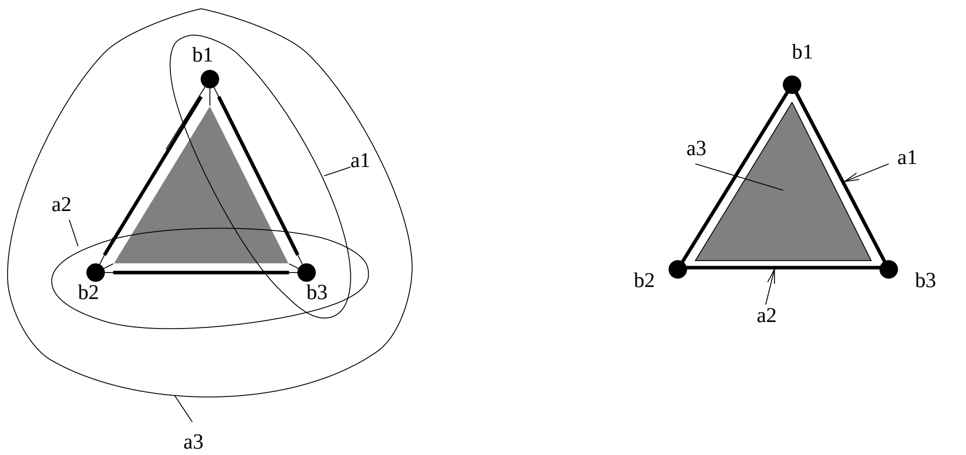
$$F = \{\text{tulipe}, \text{coquelicot}, \text{pensée}, \text{myosotis}\}$$

et l'ensemble C, de diverses couleurs

$$C = \{\text{rose}, \text{jaune}, \text{bleu}, \text{blanc}\}$$

Une relation intéressante entre ces deux ensembles serait λ telle que :

la fleur f_i est λ-reliée à la couleur c_j ssi
il existe des fleurs du type de f_i ayant la couleur c_j sur terre.



(a) Les simplexes associés à a_1 et a_2 sont des 1-simplexes, celui qui est associé à a_3, est un 2-simplexes

(b) Par convention, nous noterons le nom des 0-simplexes et des 1-simplexes à l'aide d'une flèche, celui des 2-simplexes à l'aide d'un petit rond, et plus tard, nous noterons celui des 3-simplexes à l'aide d'un petit ressort

FIG. **2.15** – Complexe associé à la matrice d'incidence d'une relation binaire. La relation représentée est telle que $\lambda = \{(a_1, b_1), (a_1, b_3), (a_2, b_2), (a_2, b_3), (a_3, b_1), (a_3, b_2), (a_3, b_3)\}$.

Comme nous l'avons vu, une manière économique de représenter λ est d'écrire la matrice d'incidence :

λ	tulipe	coquelicot	pensée	myosotis
rose	1	0	1	1
jaune	1	1	1	0
bleu	0	0	1	1
blanc	1	1	1	1

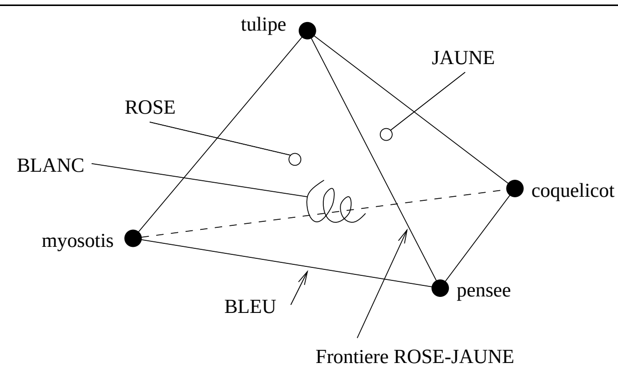


FIG. **2.16** – Complexe associé à $\lambda \subset (\boldsymbol{Fleurs} \times \boldsymbol{Couleurs})$

Nous pouvons alors représenter les couleurs comme des complexes simpliciaux, chaque couleur étant définie par une ligne de la matrice Λ et constituant un simplexe. Le 2-simplexe représentant la couleur rose sera alors :

$$rose = \langle \text{tulipe}, \text{pensée}, \text{myosotis} \rangle$$

tandis que le 2-simplexe associé à jaune sera :

$$jaune = \langle \text{tulipe}, \text{pensée}, \text{coquelicot} \rangle$$

Nous pouvons aussi nous intéresser au complexe dual où les couleurs sont des sommets, et les fleurs des simplexes (voir figure 2.17), la matrice de λ^{-1} est obtenue simplement en transposant Λ.

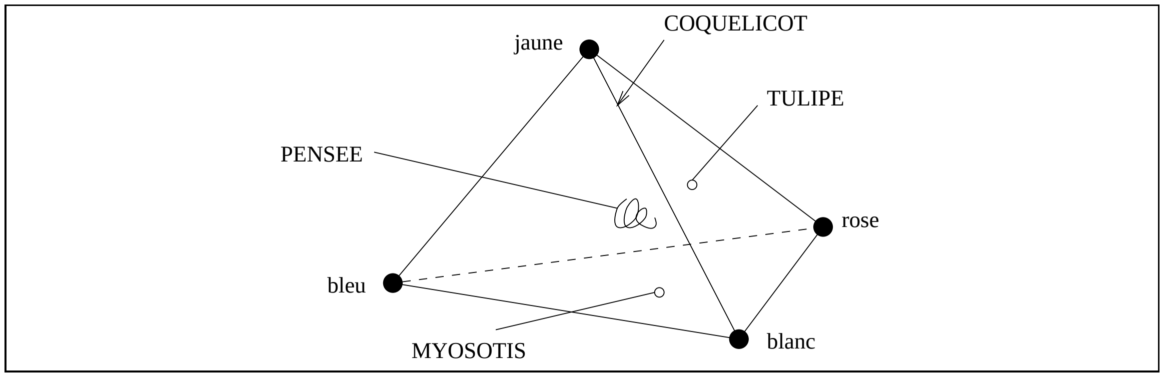


FIG. **2.17** – Complexe associé à $\lambda^{-1} \subset (Couleurs \times Fleurs)$. Les sommets sont en minuscules et sont des éléments de l'ensemble C des couleurs. Le 3-simplexe représentant l'élément *pensee* de F, contient tous les autres, il est désigné par une flèche en « ressort » .

Nous pouvons aussi exhiber, par exemple, le 1-simplexe associé à la *frontière* entre la couleur rose et la couleur jaune. On ne peut attribuer aucun nom de couleur à ce 1-simplexe, pourtant, *il ressort clairement de cette figure* et il est manipulable. Par contre, si l'on observe la table de la matrice Λ ci-dessus, les « 1» définissant cette frontière existent, certes, mais ne sautent absolument pas aux yeux.

On pourrait l'interpréter, par exemple, comme le domaine commun d'extension du rose et du jaune parmi les fleurs de la nature. Dans cet exemple, cette frontière existe d'elle-même, et correspond à un concept que nous n'avions jamais eu besoin de nommer par ailleurs. Cependant, ce type de frontière peut tout aussi bien correspondre à une sous-catégorie qui a déjà reçu un nom, car nous en avons eu l'utilité. Prenons un deuxième exemple pour illustrer cette remarque. Soit A, l'ensemble des animaux suivants :

$$A = \{\text{chameau}, \text{lézard}, \text{tigre}, \text{araignée}, \text{faisan}, \text{mésange}\}$$

et soit N, l'ensemble possible de leurs nutriments :

$$N = \{\text{eau}, \text{plante-verte}, \text{fruit}, \text{insecte}, \text{viande}\}$$

Considérons la relation :

un animal est relié à un nutriment, si ce nutriment fait partie de son régime alimentaire habituel.

Nous pouvons alors procéder à la même analyse qu'avec les fleurs, ce qui nous mène à construire un complexe composé de simplexes. Chaque simplexe représente un animal, s'appuyant sur les sommets représentant ce dont il se nourrit. Ainsi, tous les animaux « buveurs d'eau» auront un sommet $\langle \text{eau} \rangle$ commun. Tous les carnivores auront l'arête $\langle \text{eau}, \text{viande} \rangle$ en commun, les « herbivores» auront peut être l'arête $\langle \text{eau}, \text{plante-verte} \rangle$ en commun, et ainsi de suite. Nous notons que la plupart de ces frontières portent déjà un nom précis (herbivores, carnivores, ...) car elles désignent des familles d'animaux classiques.

2.5.4 Chaînes q-connectées dans un complexe simplicial

Nous présentons ici les quelques concepts introduits par R. Atkin pour analyser les propriétés d'un complexe utilisé comme représentation d'une relation binaire [Atk77].

Considérons deux simplexes σ_1 et σ_2 d'un complexe simplicial K. Nous pouvons nous intéresser à la connection qui existe entre ces deux simplexes. On dit que σ_1 et σ_2 sont connectés par une chaîne, s'il existe une séquence finie de simplexes :

$$\sigma_{a_1}, \sigma_{a_2}, \ldots, \sigma_{a_3}$$

tels que :

(1) $\sigma_{a_1} \leq \sigma_1$
(2) $\sigma_{a_n} \leq \sigma_2$
(3) σ_{a_i} et $\sigma_{a_{i+1}}$ partagent une face.

Nous parlons alors d'une *chaîne de q-connectivité*, avec q, le plus petit des entiers a_i. ainsi, un n-simplexe est n-connecté à lui-même par une chaîne de longueur 0, et ne peut être $(n+1)$-connecté à aucun simplexe. Illustrons cette notion à l'aide d'un exemple, et considérons les deux simplexes :

$$Y_1 = \langle x_1, x_2, x_3, x_4 \rangle$$

et

$$Y_2 = \langle x_2, x_4, x_5 \rangle$$

Ces deux simplexes sont 1-connectés par le 1-simplexe $\langle x_2, x_4 \rangle$ (voir figure 2.18).

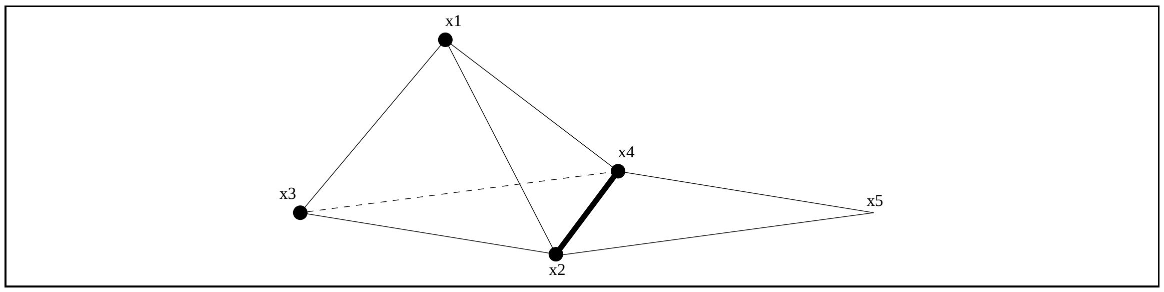


FIG. **2.18** – Un 3-simplexe et un 2-simplexe, 1-connectés le long du 1-simplexe $\langle x_2, x_4 \rangle$. Le 3-simplexe $\langle x_1, x_2, x_3, x_4 \rangle$, est en trois dimensions, l'arête en pointillés est en arrière par rapport au plan du 2-simplexe, pour figurer l'idée de volume

Pour chacun des simplexes d'un complexe K donné, nous pouvons distinguer deux valeurs particulières de q-connectivité :

- la valeur q_{min} la plus petite de connectivité du simplexe à un autre simplexe dans K ;
- la valeur q_{max} la plus grande, qui est bornée par la dimension du simplexe lui-même.

A partir de la notion de q-connectivité, nous pouvons mesurer l'*excentricité* ϵ [Atk77] :

$$\epsilon(\sigma_p) = \frac{q_{max}(\sigma_p) - q_{min}(\sigma_p)}{q_{min}(\sigma_p) + 1}$$

Cette mesure est infinie lorsque σ_p est totalement déconnecté du reste du complexe.

2.5.5 Analyse de la représentation

Le fait d'identifier les parties d'un complexe K qui sont q-connectées, pour $q \in [0, N]$ où $N = dim(K)$, réalise un partitionnement de K.

Soit une relation d'équivalence γ_q sur K, pour un q donné, définie par : $(\sigma_p, \sigma_q) \in \gamma_q$ ssi σ_p est q-connectée à σ_q. La relation γ_q est bien une relation d'équivalence puisqu'elle est réflexive, transitive et symétrique. Les classes d'équivalences sous γ_q font partie du groupe quotient $\frac{K}{\gamma_q}$ et constituent une partition de K [Atk77].

La cardinalité de $\frac{K}{\gamma_q}$, notée Q_q est égale au nombre de composantes q-connectées de K. L'analyse du complexe K consistant à trouver les nombres :

$$Q_0, Q_1, Q_2, ..., Q_N$$

où $N = dim(K)$, constitue la Q-analyse de K.

Remarquons que la valeur de Q_0 est la même que celle du 0-ième nombre de Betti[4], il donne le nombre de composantes connexes par arcs de K. Les nombres de Betti d'ordre supérieur ne correspondent pas aux autres nombres Q_i, avec $i > 0$.

2.5.6 Définition des chemins et test de la q-connectivité en Mathematica

Pour trouver un chemin dans un complexe, il faut tester si il existe des éléments communs entre les éléments deux à deux. Voici les fonctions permettant de savoir si deux simplexes sont q-connectés par un chemin de dimensions d dans un complexe c.

```
Sharing[s1_cSimplex, s2_cSimplex]:=
Intersection[CreerSimplex[s1], CreerSimplex[s2]]
Sharing[c1_cComplex,c2_cComplex]:= Intersection[Flatten[c1],Flatten[c2]]
Lconnect[s1_cSimplex,s2_cSimplex]:= Dim[Sharing[s1,s2]]
Lconnect[c1_cComplex,c2_cComplex]:= Dim[Sharing[c1,c2]]

Connections[d_Integer, s_cSimplex,c_cComplex]:=
    Select[List@@c,(d≤Dim[Intersection[s,#]])&]

CheminsQ[l_Integer,d_Integer, s1_cSimplex,s2_cSimplex,c_cComplex]:=
    CheminsQ[l-1,d,Select[List@@c,DansCQ[s1,#]&],s2,c]

CheminsQ[0,_,_,_,_]:=False
CheminsQ[_,_,{},_,_]:=False

CheminsQ[l_Integer,d_Integer,r_List,sf_cSimplex,c_cComplex]:=
    With[{voisins=Flatten[Connections[d,#,c]&/@r,1]},
        With[{target=Select[voisins,DansCQ[sf,#]&]},
            Print["voisins=",voisins];
            Print["target=",target];
            If[(target==={}),CheminsQ[l-1,d,Complement[voisins,r],sf,c],
                True]]]

CheckChemin[n_Integer,s1_cSimplex][s2_cSimplex]:=If[DansCQ[s1,s2],{n,s2},{}]

Chemins[l_Integer,d_Integer, s1_cSimplex,s2_cSimplex,c_cComplex]:=
    With[{voisins=Flatten[CheckChemin[0,s1]/@(List@@c),1]},
    With[{target=Select[voisins,DansCQ[s2,#[[2]]]&]},
            If[target==={},Chemins[0,l,d,s2,c,voisins,voisins],target]]]

Chemins[_,0,_Integer,_cSimplex,_cComplex,_List,_List]:={}
```

2.6 Représentation de prédicats avec les complexes

Nous avons adapté la Q-analyse afin de *représenter un ensemble de prédicats par un complexe.*

Un prédicat est la représentation d'une relation binaire par une fonction d'un ensemble A vers $\{0,1\}$. Nous pouvons donc analyser un prédicat comme une relation binaire incluse dans $A \times \{0,1\}$. Cependant, nous pouvons associer un complexe à un *ensemble* de prédicats, $\{p_1, p_2, ..., p_n\}$, sur A, en considérant la relation

$$\lambda = \{(a_i, p_j) \text{ tel que } p_j(a_i) = 1\}$$

Soit par exemple, A, la liste des 10 premiers entiers : $A = \{1,2,3,4,5,6,7,8,9,10\}$ et p_1, la propriété de *parité*, p_2, l'*imparité*, p_3, la *primalité*, et p_4, la propriété d'être *multiple de* 3. Alors, la

4. Voir annexe A

relation associée à A et à l'ensemble $\{p_1, p_2, p_3, p_4\}$, est :

	p_1	p_2	p_3	p_4
1	0	1	0	0
2	1	0	1	0
3	0	1	1	1
4	1	0	0	0
5	0	1	1	0
6	1	0	0	1
7	0	1	1	0
8	1	0	0	0
9	0	1	0	1
10	1	0	0	0

et on en déduit la représentation en termes de complexe de la figure 2.19. À travers ces 4 prédicats, les 10 premiers entiers sont structurés de cette manière, avec les chiffres 4, 8, 10 qui sont indiscernables.

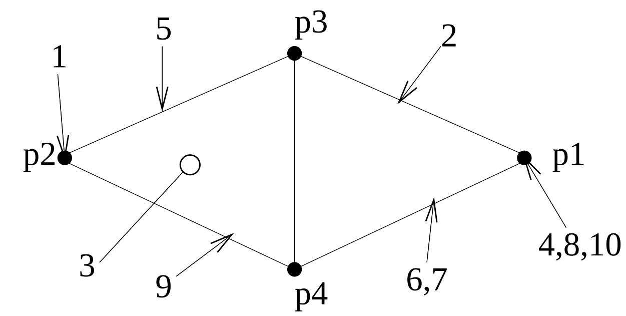


FIG. **2.19** – Complexe dual associé à $\lambda \subset \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \times \{p_1, p_2, p_3, p_4\}$

2.7 Quelques remarques sur la représentation par un complexe d'une relation binaire ou d'un prédicat

Nous aimerions développer maintenant quelques intuitions sur la représentation par des complexes, de relations binaires ou d'ensembles de prédicats.

2.7.1 Représentation par différence

La première remarque concerne le caractère *différentiel* de cette représentation. Prenons, par exemple, la représentation d'un ensemble P de prédicats sur un ensemble A d'objets. Dans une approche classique, par exemple celle des réseaux sémantiques, A et P sont des sommets, et un objet $a \in A$, vérifie un prédicat $p \in P$, s'il existe un arc entre a et p. Les ensembles A et P, et les valeurs a et p sont donnés *a priori*.

Avec une représentation à l'aide de simplexes, deux éléments ayant la même représentation, c'est-à-dire la même ligne dans la matrice d'incidence, seront confondus. Autrement dit, deux éléments de A ne sont représentés comme différents que s'il existe un prédicat (un test) de P qui permette de les distinguer. La situation est symétrique avec la représentation duale : deux prédicats ne sont représentés comme distincts que s'il existe un élément de A (un état du monde), permettant de les différencier. Cette propriété, qui peut se résumer par l'*égalité des indiscernables* (comme pour les entiers 4, 8, 10 de la figure 2.19), est en accord avec un point de vue fonctionnaliste ou nominaliste sur la représentation des connaissances.

2.7.2 Interprétation de la notion de connexité

La deuxième remarque concerne l'interprétation de la connexité dans une représentation avec des complexes. Reprenons un ensemble P, de prédicats sur un ensemble A, un élément de A étant représenté par un simplexe. Certaines faces de ce simplexe, ses ***bords***, sont partagées avec d'autres simplexes. Ainsi, un élément de A est représenté par ce qu'il a de commun avec les autres éléments, et aussi, par ce qu'il a de spécifique. Ce qu'il a en commun est vraiment partagé et permet de *cheminer* de cet élément vers les autres éléments qui partagent quelque chose avec lui.

Nous pouvons résumer notre pensée, en empruntant ces mots à L. Wittgenstein : « De même que nous ne pouvons absolument pas concevoir les objets spatiaux en dehors de l'espace ni des objets temporels en dehors du temps, nous ne pouvons imaginer ***aucun*** objet en dehors de la possibilité de sa connexion avec d'autres objets. » [Wit21]. C'est cette connexion même que nous essayons de capturer dans un CS.

Chapitre 3

Extraire et représenter des connaissances

Notre objectif dans ce chapitre est de montrer que nous pouvons utiliser les outils de la topologie algébrique (décrits au chapitre 2), pour proposer une modélisation diagrammatique (définie au chapitre 1) d'opérations cognitives variées. Nous aborderons uniquement deux types d'opérations cognitives, dans le cadre restreint d'applications précises. Nous commençons dans ce chapitre, par aborder le problème de la catégorisation pour la représentation des connaissances.

La résolution d'un problème passant par la manipulation d'une représentation, nous allons aborder tout d'abord l'extraction et la construction d'une représentation à l'aide des complexes simpliciaux.

L'extraction se fait à partir du monde environnant et grâce aux phénomènes de la perception. Holland [HHNT86] propose une petite modélisation simple de ce processus, que nous allons reprendre ici dans un but purement exploratoire. Cette modélisation présente en effet l'avantage de rester suffisamment simple.

Nous présentons une mise en œuvre informatique des propositions de Holland, à l'aide des complexes simpliciaux. Nous verrons que Holland, comme Gineste [Gin97] proposent une extraction à partir d'une catégorisation des stimuli perçus. La partition des percepts en classes d'équivalence nous permet de construire une représentation qui est en correspondance avec le monde représenté comme l'exige la définition de Rossi (voir figure 1.1 [Ros95]).

Cependant, la représentation peut être élaborée à partir des connaissances extraites et des connaissances stockées en mémoire, issues de perceptions antécédentes. Nous proposons alors, une variante, tenant compte des catégories extraites.

3.1 La représentation catégorielle

Plusieurs auteurs se sont attardés sur la question du lien qui assure le passage entre l'environnement et les modèles cognitifs. Le premier postulat énoncé dans ce domaine est celui d'isomorphisme, mais il est désormais admis ([Gin97, JL92]) que les correspondances entre les objets et les représentations mentales ne sont pas toutes isomorphes. Nous devrons donc débattre de la nature du morphisme qui lie le monde représenté au monde représentant.

Considérons pour cela, le « processus même d'ordonancement du monde des objets : la catégorisation » [Gin97]. C'est en fonction du résultat du processus de catégorisation que nous pourrons qualifier les représentations d'isomorphes ou non.

En effet, la catégorisation est un des processus fondamentaux dans l'adaptation des humains à leur environnement. Catégoriser consiste à diviser cet environnement en classes d'équivalence, c'est-à-dire à regrouper les stimuli en des ensembles selon des caractères qu'ils ont en commun. Le but est de réduire la complexité de l'environnement car cette réduction assure ensuite à l'humain une économie dans l'intégration même de l'information.

3.1.1 Qu'est-ce qu'une catégorie ?

Selon Gineste [Gin97], une catégorie est une forme particulière de représentation à laquelle un mot du lexique est associé. Pour Smith [SM81], il faut prendre en compte trois types d'objets :

- les objets du monde ;
- les noms (concepts) qui dans une langue donnée les désignent et en sont les substituts ;
- les représentations de ces mêmes objets.

Mais pourquoi introduire de façon nécessaire un nom ou une étiquette? Une catégorie peut être définie uniquement par un groupe d'objets connus sur lesquels elle *prend appui* à un moment donné. Elle peut aussi être réduite à une frontière entre deux autres catégories, il peut même ne pas y avoir de nom pour désigner toute une structure d'objets. Il sera alors difficile de mettre en relation une catégorie avec d'autres selon un seul schéma. Ces schémas seront amenés à évoluer à mesure que le système intègre de nouvelles informations.

Nous pensons que les complexes simpliciaux se prêtent tout à fait à ce genre d'exigence. En effet, ils permettent de désigner des sous-structures sur lesquelles il n'y a pas forcément la possibilité de poser une étiquette (voir par exemple, la frontière « rose-jaune » de la figure 2.16). De plus, ils offrent la possibilité de regarder une même catégorie somme faisant partie de plusieurs structures différentes, à des *échelles* différentes. En effet, un objet peut-être représenté par un p-simplexe qui participe a plusieurs relations et qui est donc pris dans plusieurs complexes, tout en étant lui-même composé de simplexes, c'est-à-dire, tout en s'appuyant sur des structures qui peuvent être vues comme étant d'une échelle inférieure. Une catégorie prise dans plusieurs complexes représentés dans des dimensions différentes, constitue un point de passage entre ces différentes structures.

Une fois représentées sur une même structure, ces différentes catégories pourront être manipulées grâce aux outils de la topologie algébrique tout en gardant la possibilité de passer naturellement d'une structure à laquelle elle participe, à une autre.

3.1.2 Processus d'extraction

Nous ne rentrerons pas dans le détail des processus même de l'extraction qui relèvent de la biologie et surtout de la psychologie expérimentale. Cependant, nous voudrions faire quelques remarques sur les contraintes dues aux processus d'extraction et de catégorisation.

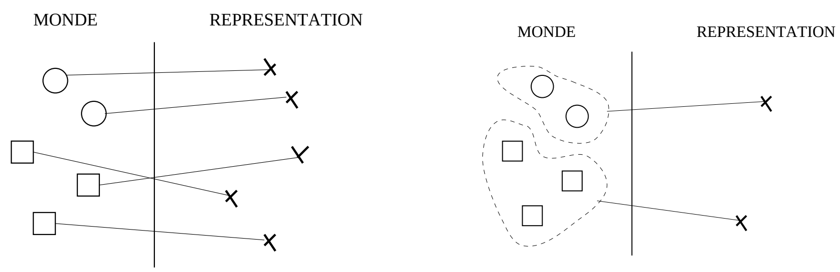
La classification peut se faire selon plusieurs processus d'identification, les propriétés relevées ne sont en particulier pas obligatoirement « objectives » . Ainsi, il est important de noter que la catégorisation à laquelle aboutira un système cognitif à un instant donné dépend, entre autres, de l'ordre dans lequel il aura perçu les objets du monde. Ceci implique qu'aucune taxonomie des types ne peut être décidée *a priori*, une représentation est extrêmement dépendante du système cognitif observé et du processus d'observation.

À l'heure actuelle, nous retiendrons les contraintes de relativité et de dynamicité comme principales. C'est pour cela qu'il convient de choisir un outil qui privilégie les relations entre les catégories et qui se caractérise par leur configuration relative. Encore une fois, les complexes simpliciaux nous paraissent satisfaire à cette double exigence.

3.2 Quel morphisme pour les représentations ?

Dans le cas d'une correspondance terme à terme on pourrait parler d'isomorphisme entre les modèles cognitifs et l'environnement (voir figure 3.1(a)).

Étant données la complexité du monde environnant et les limitations des systèmes cognitifs réalistes, il est cependant déraisonnable de postuler que les modèles mentaux sont isomorphes. Nous allons voir quel morphisme est pertinent dans l'hypothèse d' une représentation catégorielle.



(a) Isomorphisme, correspondance entre le monde représenté et le monde représentant$(1-1)$

(b) Homomorphisme, correspondance entre le monde représenté et le monde représentant de $(n-1)$

FIG. **3.1** – Un isomorphisme et un homomorphisme entre le monde et sa représentation par un système cognitif.

3.2.1 L'homomorphisme pour la représentation catégorielle

En général, un système cognitif essaiera de construire plusieurs modèles simplifiés pour atteindre ses buts, en ignorant certains détails. Nous avons vu qu'un des processus importants d'extraction des connaissances pour élaborer cette représentation est la catégorisation. Il existe plusieurs approches de l'activité de catégorisation. Holland [HHNT86] en propose une, basée sur l'existence d'une fonction simple de catégorisation, réalisant des classes d'équivalence sur les stimuli. Cette proposition est illustrée par la figure 3.2.

Dans ce modèle, les détecteurs d'un système cognitif sont modélisés simplement comme pouvant prendre uniquement les valeurs binaires *allumé* ou *éteint* (0 ou 1), ce sont donc des implémentations physiques de prédicats. Soient d_1, d_2, d_3, quatre de ces détecteurs, ils codent des propriétés de l'état du monde, et une fonction de catégorisation C, définie à partir des propriétés détectées, partitionne l'ensemble des états du système. Ainsi les détecteurs divisent le monde en des classes d'équivalence, traitant les objets que les détecteurs ne permettent pas de distinguer comme équivalents.

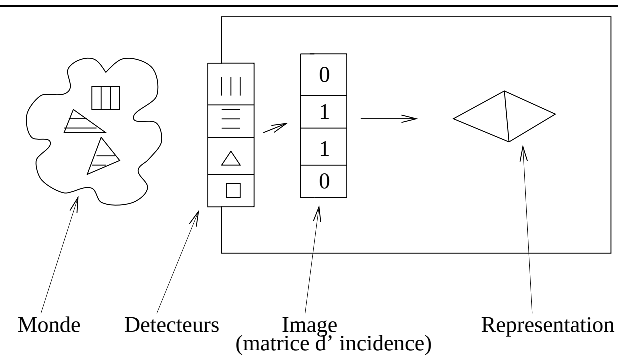


FIG. **3.2** – Équivalence entre un état du monde et une représentation : les détecteurs s'allument selon les propriétés des objets observés. Les deux triangles rayés horizontalement sont perçus comme équivalents par ce système qui ne discerne pas les tailles. L´état des détecteurs va permettre ensuite l'élaboration d'une représentation.

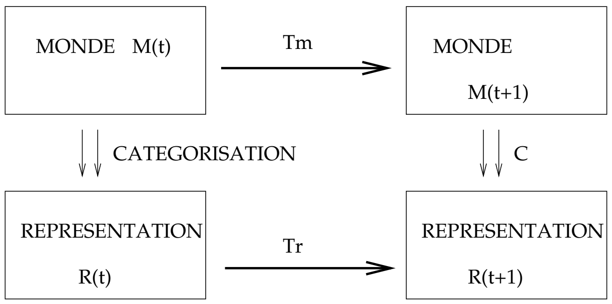
L'encodage d'un objet correspond alors à l'application, sur lui, des prédicats physiquement implémentées par les détecteurs. Par exemple, un triangle rayé horizontalement sera un objet sur lequel les prédicats *triangle* et *rayures horizontales* se seront appliquées. Un modèle du monde basé sur une telle catégorisation met en correspondance plusieurs objets du monde avec une classe du modèle. Cette correspondance $n \to 1$ est un homomorphisme (voir figure 3.1(b)).

3.2.2 Prise en compte des exceptions, quasi-homomorphisme

Supposons maintenant, que le temps est discret, et que le monde M représenté est dans l'état $M(t)$ à l'instant t, et qu'il sera dans l'état $M(t+1)$ à l'instant suivant.

Nous introduisons la notion de fonction de transition pour décrire le passage du monde entre l'état $M(t)$ et $M(t+1)$, appelons cette fonction T_m. On aura :

$$M(t+1) = T_m(M(t))$$



TAB. 3.1 – *Fonction de transition du monde représenté et du monde représentant : homomorphisme*

Soit $R(t)$, la représentation de l'état $M(t)$ du monde, à l'instant t. Le but pour le système cognitif lorsqu'il se crée une représentation est d'effectuer des prédictions valables sur son environnement. Il va donc lui falloir une représentation de la fonction T_m, que l'on va appeler T_r. Un modèle du monde sera valide s'il permet d'effectuer les bonnes prédictions, c'est à dire que le fait d'appliquer T_r à l'état du monde représentant $R(t)$ doit mener à un état $R(t+1)$ qui soit la représentation du monde $M(t+1)$, en d'autres termes, avec :

$$R(t+1) = T_r(R(t))$$
$$R(t) = C(M(t))$$

$R(t+1)$ doit satisfaire à la relation :

$$R(t+1) = C(M(t+1))$$

ce qui permet de déduire que :

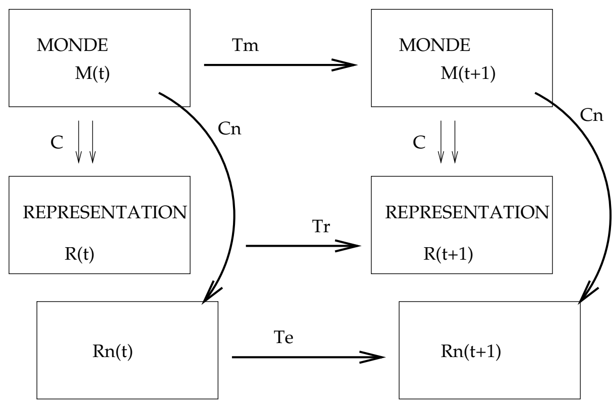
$$C(T_m(M(t))) = T_r(C(M(t)))$$

Autrement dit, il faut qu'il y ait « commutativité » entre la fonction de catégorisation C, et le passage d'un état de l'instant t à l'instant $(t+1)$.

L'exemple pris ci-dessus pour illustrer la relation d'homomorphisme entre l'environnement et les modèles mentaux paraît fort simple comparativement à la complexité de l'environnement. On peut par exemple envisager que certains objets appartiennent à une classe tout en différant par certaines propriétés, la catégorisation mérite alors d'être affinée.

En effet, il se peut que certains événements n'obéissent pas aux prédictions faites par le modèles, c'est alors une *exception*. Ces exceptions peuvent être gérées en ajustant la représentation, qui devient $R_n(t)$ de la catégorie concernée (représentation dynamique) et donc en ajustant la fonction de transition T_r, qui devient T_n. Ceci est schématisé sur la figure 3.2, sur laquelle on voit que la transformation se fait en deux étapes, et que la fonction de catégorisation est également ajustée en C_n. L'existence de ces exceptions ne nous permet plus de parler d'homomorphisme, on a alors seulement un quasi-homomorphisme [HHNT86].

Le schéma 3.2 est à retenir, car il nous amène naturellement à parler de raisonnement par analogie dans le chapitre suivant. En effet, les représentations T_r et T_n sont construites par analogie avec la relation T_m. De plus, Holland s'intéresse lui-aussi au raisonnement par analogie, et ses collègues Thagard et Holyoak ont développé un simulateur pour le raisonnement analogique [1], ACME. Nous nous intéresserons au raisonnement par analogie au chapitre suivant.



TAB. 3.2 – *Gestion d'une exception par affinement du modèle : quasi-homomorphisme*

3.3 Une application simple d'extraction

A partir de ce que nous venons de voir, nous allons essayer de mettre en œuvre une petite simulation de ce modèle pour l'extraction. Évidemment, ce que nous voulons extraire, est un complexe simplicial. Nous allons choisir un nombre fini de capteurs discrets pouvant prendre la valeur *allumé* ou *éteint*. Nous pouvons par exemple imaginer qu'il s'agit d'une rétine munie de plusieurs capteurs qui forment une matrice, mais la forme et la nature précise de ces capteurs n'a pas d'importance pour ce qui nous préoccupe.

3.3.1 Hypothèses et analyse du problème

Nous devons préciser que notre ambition n'est absolument pas de faire de la reconnaissance des formes exacte. Nous nous intéressons seulement à la faisabilité conceptuelle des thèses de Holland, en utilisant des complexes simpliciaux pour représenter les connaissances extraites.

Soit un système cognitif minimal, *i.e.* qui est physiquement capable de découper dans le donné ce que nous appelerons des *images*. Une *image* est un ensemble non ordonné d´eléments atomiques différents les uns des autres. On peut se représenter une telle image, de manière shématique, par une liste non ordonnée d'entiers tous différents. En fait, nous ne nous intéressons pas à l'ordre sur les entiers, ils ne servent que d'étiquettes pour repérer les percets atomiques.

Le système est donc supposé capable d'un découpage selon deux modes :

- dans l'espace : nous obtenons alors un ensemble non ordonné d'images ;
- dans le temps : nous obtenons alors un ensemble ordonné d'images.

Dès la détection, la fonction de catégorisation réalise la correspondance entre les états du monde et leur représentation. Deux objets qui ne sont pas différenciables grâce aux détecteurs en question feront partie de la même classe d'équivalence pour notre système cognitif.

Nous allons simuler cette extraction selon deux processus, correspondant aux deux modes de découpage décrits ci-dessus, nous pourrons ensuite les comparer. Il est à souligner que l'étape clé de l'extraction se situe au niveau de cette heuristique. Bien sûr, il faudrait en explorer plusieurs en ayant soin de leur vraisemblance sur un plan psychologique.

1. Ce point est développé dans le chapitre 4, et le simulateur ACME est présenté. Thagard et Holyoak sont les co-auteurs du livre présentant le modèle de Holland [HHNT86] dont nous nous inspirons ici.

Cependant, pour notre approche naïve de ce problème nous nous limiterons aux deux processus suivants :

- le premier extrait une base à partir d'un découpage dans l'espace, donné *a priori* ;
- le deuxième, dit incrémental, extrait une base au fur et à mesure que des images composées du monde lui sont présentées, c'est-à-dire qu'il procède par de'coupage dans le temps.

Pour construire une ontologie sur la base des concepts extraits, nous nous intéressons aux éléments primitifs qui suffisent pour décrire l'univers, on cherchera donc à constituer une *base*, et à décrire chaque état du monde comme une composition des éléments de la base. Nous entendons par *base*, un ensemble de simplexes disjoints qui permettent d'exprimer les images perçues en les combinant selon un processus qui est proche d'une combinaison linéaire.

Les deux heuristiques d'extraction peuvent se situer à deux extrémités d'un axe nécessitant le développement d'approches intermédiaires par la suite. Intuitivement, le premier processus revient à comparer entre tous les états du monde qui ont été perçus, tandis que le deuxième processus tient compte du passage du temps et de la présentation séquentielle des perceptions.

3.3.2 Description de l'implémentation

Dans cet exemple, nous désignons un stimulus par un *nombre entier*. Deux stimulus sont différents s'ils sont représentés par des nombres différents. Nous ne nous intéressons pas aux relations d'ordre qui existent par ailleurs entre les nombres. L'ensemble des stimuli qui se présentent à un instant t donné au système, est représenté par une liste de nombres. L'ordre des éléments dans cette liste n'a pas d'importance. Une suite de perceptions, est une suite de ces listes.

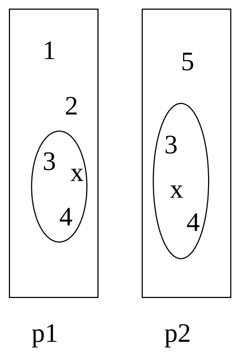
Pour extraire les objets élémentaires, à partir de ces perceptions, il faut au moins deux perceptions successives. Soient $P_1 = \{1, 2, 3, 4\}$, et $P_2 = \{3, 4, 5\}$, deux perceptions.

L'intersection de P_1 et P_2 est $\{3, 4\}$, elle peut correspondre à :

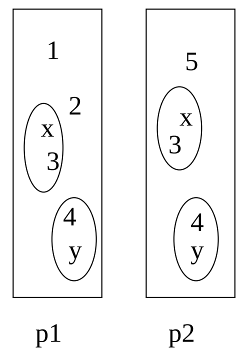
1. Un seul objet du monde qui présente les attributs 3 et 4, et qui est présent une fois dans chaque perception (voir figure 3.3(a)).
2. Deux objets différents, l'un présentant l'attribut 3, l'autre, l'attribut 4, présents à chaque fois dans les deux perceptions (voir figure 3.3(b)).
3. Des attributs communs à deux objets différents dans chaque perception, par exemple ici, l'objet $\{1, 3, 4\}$ dans P_1, et l'objet $\{3, 4, 5\}$ dans P_2 (voir figure 3.3(c)).
4. Des attributs d'objets différents dans une perception, et différents aussi entre les perceptions. Par exemple, $\{1, 3\}$, et $\{2, 4\}$, dans P_1, et $\{3, 4\}$ et $\{4, 5\}$ dans P_2 (voir figure 3.3(d)).

Ainsi, l'intersection des deux perceptions ne correspond pas obligatoirement à un seul objet. Cependant, s'il y a un objet en commun entre les deux perceptions, il est au moins inclus dans l'intersection de P_1 et P_2. Nous pouvons faire un raisonnement analogue pour la différence $P_1 - P_2$ et $P_2 - P_1$. Nous calculons donc toutes les intersections et les différences de perceptions, ceci est effectué par la fonction `Fragments` ci-dessous. La fonction `Fragments` appelle une autre fonction, `Diff`, qui est chargée de calculer l'intersection et les différences.

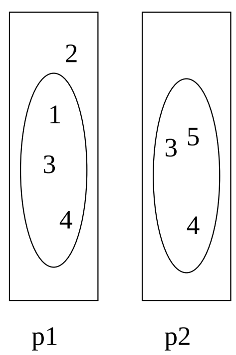
```
Diff[l1_,l2_] :=
    With[{inter=Intersection[l1,l2]},
        {inter,Complement[l1,inter],Complement[l2,inter]}]

Fragments[l_List] :=
    With[{l2 = Flatten[Outer[Diff, l, l, 1], 2]},
    With[{l3 = Select[l2,(# =!= {})&]},
        Union[l3]]]
```



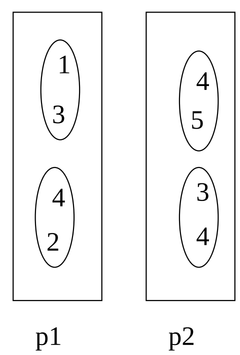
(a) Cas $n^o 1$:un seul objet du monde qui présente les attributs 3 et 4, et qui est présent une fois dans chaque perception, par exemple ici $\{3, 4, x\}$



(b) Cas $n^o 2$: deux objets différents, l'un présentant l'attribut 3, l'autre, l'attribut 4, présents à chaque fois dans les deux perceptions



(c) Cas $n^o 3$: des attributs communs à deux objets différents dans chaque perception, par exemple ici, l'objet $\{1, 3, 4\}$ dans P_1, et l'objet $\{3, 4, 5\}$ dans P_2



(d) Cas $n^o 4$: des attributs d'objets différents dans une perception, et différents aussi entre les perceptions. Par exemple, $\{1, 3\}$, et $\{2, 4\}$, dans P_1, et $\{3, 4\}$ et $\{4, 5\}$ dans P_2

FIG. **3.3** – Configurations possibles de perceptions donnant $p_1 \cup p_2 = \{3, 4\}$

Pour trouver quels sont les *objets* parmi ces intersections, nous cherchons les éléments les plus élémentaires, permettant d'exprimer toutes les autres intersections, c'est-à-dire que nous cherchons une *base*.

La fonction `SimplexBase`, renvoie un simplexe construit à partir de l'extraction d'une telle base. La base est construite au terme d'une perception globale, puisque ous sommes dans le cas d'un découpage dans l'espace.

```
SimplexBase[l_List] := (CreerSimplex@@#)& /@ SimplexBase[Fragments[l],{}]

SimplexBase[{},r_List] := r
SimplexBase[{x_,l___}, r_List] :=
    With[{reste = Complement[x,Sequence@@r]},
        SimplexBase[{l}, If[reste==={},r,Join[r,{reste}]]]]
```

Pour modéliser une extraction incrémentale, comme nous l'avons proposé, nous devons permettre au système de constituer sa base au fur et à mesure des perceptions. Nous introduisons pour cela la fonction `Incremente`, qui met à jour a base selon un processus différent. Si une nouvelle image perçue n'est pas exprimable à l'aide de la base courrante, les parties non exprimables sont alors ajoutées à la base.

```
Coord[base_List][x_List] :=
    CreerComplex@@Select[base, (Complement[Sommets[#], x]==={})&]

DecLineaire[base_List][x_List] :=
  Select[base, (Complement[Sommets[#], x]==={})&]
ClearAll[Ajoute, Incremente]
Ajoute[base_List, image_List] :=
    With[{decomp = Sommets/@(DecLineaire[base][image])},
     With[{projo=  decomp },
        With[{ reste= Complement[image,Union[Join@@projo]]},
            If[reste==={},base,Join[base,{CreerSimplex[reste]}]]]]]

Incremente[hist_List] := Fold[Ajoute,{},hist]
```

Cette fois, la base est construite en ajoutant entièrement les éléments qui ne peuvent pas être exprimés comme une décomposition linéaire des éléments déjà présents dans la base. Cette décomposition linéaire est effectuée par la fonction `DecLineaire`.

3.3.3 Comparaison des deux heuristiques d'extraction

Soit la liste arbitraire de perceptions `l`, une perception étant notée entre $\{\}$:

```
l = {{1, 2, 3}, {3, 4}, {5, 6}, {1},{1, 2, 3, 4}, {3, 4, 5, 6}, {1, 2, 3}};
```

La base extraite par le système, si nous lui demandons de le faire par la méthode incrémentale, est donnée par :

```
Incremente[l]

↪ {cSimplex[{1, 2, 3}, void], cSimplex[{3, 4}, void],
     cSimplex[{5, 6}, void], cSimplex[{1}, void]}
```

Cette base est constituée des premiers éléments perçus pris dans l'ordre, jusqu'à ce qu'un élément puisse être exprimé en fonction des stimuli déjà inclus dans la base. Ceci est dû au fait que l'extraction se fait incrémentalement, mais sans révision de la base à chaque étape, pour aboutir à une base optimale.

Ainsi, lorsque l'ensemble $\{1, 2, 3\}$ est perçu, la base étant vide, elle ne permet pas de l'exprimer, et c'est l'élément tout entier qui est inclus dans la base. L'élément suivant, $\{3, 4\}$ ne peut pas non plus être exprimé par la base $\{1, 2, 3\}$, il doit donc lui-aussi être entièrement intégré dans la base. Ainsi de suite jusqu'à ce que l'élément $\{1, 2, 3, 4\}$ soit exprimable comme combinaison de $\{1, 2, 3\}$ et $\{3, 4\}$. Les éléments suivants, sont eux-aussi exprimables à l'aide de cette base, qui n'a donc plus de raison de changer.

Observons maintenant l'extraction provenant d'une perception groupée de toute la liste de stimuli, en appliquant la fonction `SimplexeBase` aux `Fragments` issus de `l` :

```
Fragments[l]
base=SimplexBase[l]
↪ {cSimplex[{1}, void], cSimplex[{2}, void], cSimplex[{3}, void], cSimplex[{4},
    void], cSimplex[{5, 6}, void]}
```

Cette liste de `cSimplex` est bien une base. Le stimulus $\{5, 6\}$, ne sont jamais dissociés dans aucune perception, les autres, si.

3.4 Application avec sémantique de plus haut niveau

3.4.1 Hypothèses et analyse

Nous supposons que par le même processus décrit précédemment, nous pouvons construire des concepts de plus en plus élaborés, empilés sur les concepts extraits de bas niveau. Nous proposons une extraction sur le même principe, mais avec une sémantique liée aux stimuli, plus importante. Pour cela, nous avons choisi de représenter les éléments d'un conte, à savoir « le petit chaperon rouge ».

Les perceptions sont maintenant des images, ou des scènes du conte, et la liste de perceptions, est maintenant l'ordre des scènes dans le conte, son déroulement dans le temps. Ainsi, nous appelons `cr` (pour Chaperon Rouge), la liste des perceptions, et `Image`, une perception. Les stimuli ne seront plus représentés à l'aide de nombres mais de mots désignant des concepts étant supposés déjà élaborés dans le passé du système cognitif.

Nous ne nous attacherons pas à la sémantique contenue dans ces mots, mais à la structure relative des concepts, qu'ils exhibent par leur occurrence dans l'histoire. Lorsque nous parlons de sémantique de plus haut niveau, nous parlons de la sémantique du conte pris comme un tout et se déroulant dans le temps.

Nous pouvons extraire différentes sortes de structures du conte : la cohérence du monde dans lequel le conte est plongé, ou bien la structure due à la succession d'images. Ainsi, nous pouvons construire une représentation sous divers angles à partir des mêmes informations :

- structurer les objets du conte comme sous-ensemble des objets du monde et refléter une sous-structure caractéristique de la cohérence du conte vis à vis du monde dans lequel il se déroule ;
- structurer les occurrences d'objets entre eux : quelles sont les associations occurentes, c'est là que réside un « certain sens » ;
- structurer la succession des occurrences des objets dans la succession des images du conte, c'est la structure du « fil » du conte, que l'on pourra comparer à d'autres contes.

D'autres analyses sont possibles. Dans l'exposé qui suit, nous nous intéressons uniquement à la catégorisation des objets du conte.

3.4.2 Extraction de catégories du « Petit Chaperon Rouge »

Nous avons codé les images successives du conte en termes d'éléments présents ou non dans la scène. Nous avons ramené la totalité du conte à une dizaine de scènes clé, nommées `image1` à `image11` dans le déroulement de l'histoire et à 4 personnages.

Le codage est fait en fonction des objets, des personnages, des actions reconnues par le système, *sans ordre* particulier. Par exemple, dans la scène `image1`, la mère parle au Chaperon Rouge, il y a donc un `Adulte`, un `Chaperon`, dans une `maison`, en train de `parler`. Les concepts sont supposés perçus et reconnus sous cette forme par le système cognitif. Le codage des douze scènes principales du conte est alors le suivant :

```
image1 ={"Chaperon","Adulte", "parler", "maison"};
image2 ={"Chaperon","Adulte", "donner","panier", "maison"};
image3 ={"Chaperon", "marcher", "arbres","panier"};
image4 = {"Chaperon", "Loup", "parler", "arbres","panier"};
image5 ={"Chaperon", "marcher", "arbres","panier"};
image6 ={"Loup", "marcher", "arbres"};
image7 = {"Adulte","dormir","maison","lit"};
image8 = {"Adulte", "Loup", "parler","maison","lit"};
image9 ={"Adulte", "Loup", "manger","maison","lit"};
image10 ={"Chaperon", "parler","maison","panier","lit"};
image11 ={"Chaperon", "Loup", "manger","maison","panier","lit"};
```

Par exemple, l'image `image6`, correspond au loup qui court pour aller chez la grand-mère, la scène `image10` correspond à l'arrivée du chaperon après que le loup ait mangé la grand-mère. Dans cette version originale du conte, le chasseur ne vient délivrer personne à la fin.

La succession des images est appelée `cr`, et correspond au déroulement du récit dans le temps :

```
cr = {image1, image2, image3, image4,image5, image6, image7, image8, image9,
image10, image11};
```

Rappelons que l'ordre des éléments dans une image n'est pas significatif, mais la succession des images est significative pour le processus d'extraction incrémental.

La base d'objets, d'actions et de personnages à partir de laquelle nous avons raconté l'histoire est donnée par :

```
percepts = Union[Flatten[cr]]

↪ {"Adulte","arbres","Chaperon","donner","dormir","lit","Loup","maison",
   "manger","marcher","panier","parler"}
```

Voyons quelle est la base extraite par le système de manière non incrémentale, c'est-à-dire sans la connaissance *a priori* de notre façon de la raconter :

```
SimplexBase[cr]

↪ {cSimplex[{"Adulte"},void],cSimplex[{"Chaperon"},void],
   cSimplex[{"dormir"},void],cSimplex[{"Loup"},void],cSimplex[{"maison"},void],
   cSimplex[{"manger"},void],cSimplex[{"marcher"},void],
   cSimplex[{"parler"},void],cSimplex[{"donner"},void],
   cSimplex[{"arbres"},void],cSimplex[{"panier"},void],cSimplex[{"lit"},void]}
```

Les éléments de la base calculés, sont exactement ceux de l'ensemble de percepts de base que nous nous sommes donnés pour décrire le conte. Ceci est dû à notre façon de décrire les scènes du conte. En effet, il y a peu de stimuli différents par image, et il y a suffisamment d'images pour exhiber chaque stimulus seul au moins une fois dans l'ensemble des intersections.

Maintenant, introduisons l'idée de temps dans le récit, et utilisons l'heuristique incrémentale. Pour extraire une base incrémentalement, nous ajoutons une première perception, qui est notée `image0` :

```
image0={"Adulte", "maison"};
```

En effet, nous avons vu précédemment, que le premier percept entre tel quel dans la base puisqu'elle ne contient rien au début. Ceci contraint la suite des percepts a être très peu morcelée, car elle est difficilement exprimable en termes d'un objet trop compliqué. Cette première image constitue un percept simple, qui nous permet de contourner ce biais, en imposant une base pas tout à fait vide au début du conte, ce qui correspond de plus à une certaine réalité du système cognitif. La base d'un système capable de reconnaître des concepts comme `parler`, ne sera jamais vide, elle sera composée d'éléments suffisamment petits pour morceler le conte.

Extrayons une base à partir de la succession d'images, `cr`, de manière incrémentale, voici la base que le système propose :

```
Incremente[cr]

↪ {cSimplex[{"Adulte","maison"},void],cSimplex[{"Chaperon","parler"},void],
   cSimplex[{"Chaperon","donner","panier"},void],
   cSimplex[{"arbres","Chaperon","marcher","panier"},void],
   cSimplex[{"arbres","Loup","panier"},void],
   cSimplex[{"arbres","Loup","marcher"},void],cSimplex[{"dormir","lit"},void],
   cSimplex[{"lit","Loup","parler"},void],
   cSimplex[{"lit","Loup","manger"},void],
   cSimplex[{"lit","maison","panier"},void],cSimplex[{"Chaperon"},void]}
```

On obtient, cette fois, des « pattern-types » comme par exemple `{dormir, lit}`. Mais recommençons le codage du conte, rendons-le plus fin pour permettre une création d'une base un peu moins calquée sur l'ensemble des percepts de base. Pour ce faire, nous augmenterons le nombre de percepts par images, ceci pour un nombre constant d'images.

```
codage1= {
"Adulte" -> {"vivant", "agréable"},
"arbres" ->{"vivant", "lieu", "méchant", "plusieurs"},
"Chaperon"->{"vivant", "agréable","petit"},
"donner"->{"moteur", ".->"},
"dormir" ->{"moteur","agréable"},
"lit" -> {"lieu", "agréable"},
"Loup" -> {"vivant","animal","petit"},
"maison"-> {"lieu"},
"manger"->{"moteur", ".<-", "agréable"},
"marcher"->{"moteur"},
"panier"->{"petit"},
"parler"->{"moteur", ".->"}
};
```

Le conte est alors donné par la succession des images, dans lesquelles on a remplacé les anciens stimuli par les nouveaux codages :

```
cr1 = Flatten /@(cr /.  codage);
```

La base incrémentalement extraite du nouveau conte perçu, `cr1`, est alors :

```
b1=Incremente[cr1]

↪ {cSimplex[{"agréable","lieu","moteur","petit","vivant",".->"},void],
   cSimplex[{"agréable","lieu","méchant","moteur","petit","plusieurs",
   "vivant"},void],cSimplex[{"animal"},void],
   cSimplex[{"lieu","méchant","moteur","petit","plusieurs","vivant"},void],
   cSimplex[{"agréable","lieu","moteur","vivant"},void],
   cSimplex[{"petit",".<-"},void]}
```

Tandis que la base extraite instantanément de `cr1`, toujours sans la scène `image0`, est :

```
b2=SimplexBase[cr1]

↪ {cSimplex[{"agréable"},void],cSimplex[{"animal"},void],
   cSimplex[{".->"},void],cSimplex[{".<-"},void],
   cSimplex[{"méchant","plusieurs"},void],cSimplex[{"petit"},void],
   cSimplex[{"lieu","moteur","vivant"},void]}
```

La représentation de cette dernière base à l'aide d'un diagramme de Hasse (voir figure 3.4), montre que le groupe d'attributs le plus grand de cette base est `{lieu, moteur, vivant}`, qui peut s'interpréter comme une nouvelle catégorie.

La représentation sous forme de diagramme de Hasse de la base permet de visualiser de manière particulièrement bien adaptée l'ontologie des catégories extraites du conte. Au rez-de-chaussée du diagramme, on trouve naturellement les catégories correspondant à l'allumage d'un seul capteur. Si un tel 0-simplexe apparaît dans la base, cela veut dire qu'il existe des états du monde permettant de distinguer cette perception pure.

Les catégories d'un étage n, sont « ontologiquement premières » par rapport à celles de l'étage $(n+1)$. Illustrons cela par l'exemple classique de l'espace et de la couleur : on dit que l'étendue est une catégorie qui précède ontologiquement la couleur, car on ne peut pas percevoir, imaginer ou concevoir de couleur sans percevoir, imager ou concevoir une étendue (colorée). Cette structure causale de l'ontologie est rendue par le diagramme de Hasse. Dans notre exemple, la catégorie extraite $\langle \text{lieu}, \text{moteur}, \text{vivant} \rangle$, est seconde par rapport à la catégorie $\langle \text{vivant} \rangle$.

3.4.3 Discussion des résultats

Nous allons tenter de commenter ces résultats, puisque nous ne pouvons pas nous empêcher de les interpréter. Cependant, cette analyse a surtout pour but, de convaincre le lecteur, de l'intérêt

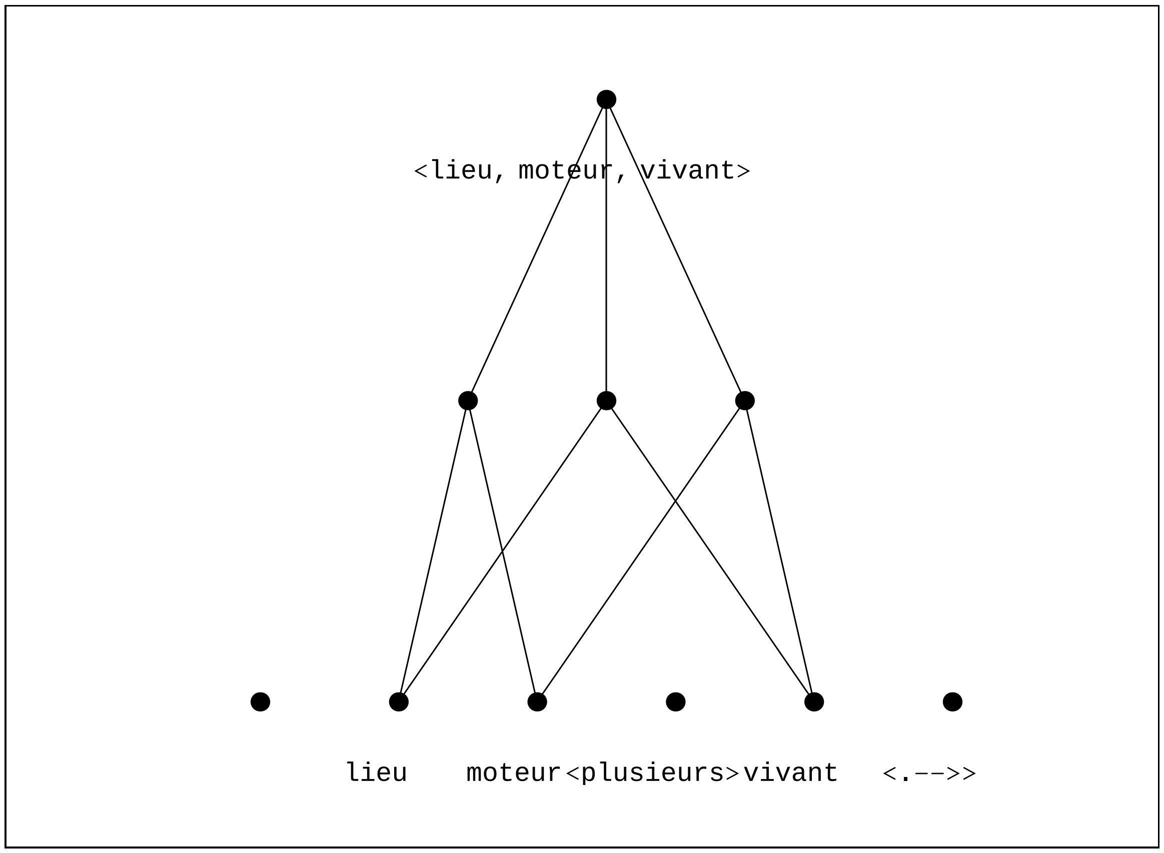


FIG. **3.4** – Morceau de base extraite de la description du Petit Chaperon Rouge en 12 scènes codées à l'aide du deuxième codage, par une extraction incrémentale que peut offrir ce type d'approche pour une analyse narratologique (automatique, par exemple). Nous insistons une dernière fois, sur la simplicité de nos hypothèse, de nos résultats, et de l'analyse qui va suivre.

Les concepts représentés par des simplexes, et que l'on peut intuitivement associer à chacune de ces bases ne portent pas obligatoirement de nom dans notre façon habituelle de raconter le conte du petit chaperon rouge. Pourtant, on peut en déduire, qu'il existe une association d'attributs qui n'est jamais décomposée en plus petits éléments dans le conte. Du moins, dans les scènes importantes du conte, que nous avons sélectionnées.

En effet, la base élaborée instantanément contient des groupements d'attributs que le système n'a jamais réussi à perçevoir de façon dissociée. N'oublions pas que la base extraite instantanément ne comprend pas le percept `image0`.

Parmi ces groupements, examinons, par exemple, `{méchant, plusieurs}`. Nous voyons que ces attributs ont été utilisés pour désigner les arbres de la forêt et nulle part ailleurs. Ainsi, il est normal, que ni `plusieurs`, ni `méchant`, n'apparaisse combinés à d'autres stimuli sans être accompagnés l'un de l'autre.

Considérons le groupe `{lieu, moteur, vivant}`. Nous n'avons pas utilisé les attributs `lieu` et `moteur`, ensembles pour définir un des éléments du conte. L'attribut `lieu`, est inclus dans les descriptions des `arbres` et du `lit`, l'attribut `moteur`, est utilisé dans les descriptions de `donner`, `dormir`, `manger`, `marcher` et `parler`.

Pourtant, d'après la base extraite instantanément par le système, ces attributs sont indiscernables dans sa perception du conte. Cela signifie qu'ils apparaissent toujours ensemble, peut-être parce qu'une `Image` est toujours décrite avec un cadre, défini par un `lieu`, et avec une action qui s'y déroule, décrite par `moteur` (sauf pour la scène `image0`, qui est exclue ici).

Le « concept » d'action-lieu est donc toujours présent et indissociable à chaque scène du conte. Ce qui n'est pas surprenant, puisque nous avons décrit le conte à l'aide des principales scènes, c'est-à-dire, celles où il se passe quelque chose, quelque part.

Mais l'analyse du groupe `{lieu, moteur, vivant}` n'est pas terminée. En effet, l'attribut `vivant` est lui aussi, toujours associé au `lieu` et aux actions décrites par `moteur`. L'attribut `vivant` est utilisé dans la description des personnages du conte `Adulte`, `Chaperon` et `Loup`, mais aussi pour la description des `arbres`. Or, nous savons que chaque scène du conte est aussi décrite en termes de personnages qui accomplissent une action, il est donc normal que l'attribut commun aux personnages apparaisse dans ce groupe qui contient les attributs présents dans toutes les descriptions. Nous aboutissons donc au résultat suivant : toutes les scènes sont décrites en termes de personnage(s), agissant dans un lieu.

Encore une fois, ceci ne nous surprend pas, car nous avons de la connaissance *a priori* sur ce que doit être un conte ou une histoire. De plus nous avons biaisé la description du conte en ne racontant que les scènes dites « importantes » , ce qui revient à décrire les scènes d'action. Aucune scène purement descriptive de la maison de la grand-mère, ou de son lit, ne sont fournies. Le fait de savoir si de telles scènes sont pertinentes ou non dans un récit, sort du cadre de notre étude.

Cependant, nous avons exhibé une combinaison, une structure qui apparaît à chaque scène, sans forcément l'analyser avec notre connaissance extérieure, nous avons un « pattern » nécessaire à la description de toute scène de ce conte, ou d'autres contes.

Ainsi, il serait intéressant de pouvoir comparer la structure du Petit Chaperon Rouge avec celle d'un autre récit du même genre, un autre conte, par exemple Pierre et Le Loup.

Les structures intéressantes à comparer sont de diverses natures. En effet, nous avons mentionné précédemment les différentes structures du Petit Chaperon Rouge que nous aurions pu essayer d'extraire. *Ici, nous nous sommes intéressé presque exclusivement au contenu commun des scènes du conte.* À partir de cette étude, nous avons mis en évidence un « pattern » caractéristique des scènes. Nous pouvons comparer ce « pattern » à celui de Pierre et le Loup, s'il y en a un.

Mais nous pourrons également comparer leurs structures temporelles, sans doute y a-t-il aussi des patterns caractéristiques du déroulement d'un récit de ce genre. Nous pourrions également procéder à une analyse de la hiérarchie des personnages [2] : si le personnage est important, il pourra apparaître seul ou avec d'autres et rencontrer la plupart des autres protagonistes.

2. En narratologie, la hiérarchie des personnages permet d'analyser un récit, cela constitue le critère d'autonomie différentielle

Chapitre 4

Modélisation d'un raisonnement par analogie : le système ESQIMO

Dans ce chapitre, nous allons étudier une mise en œuvre de plus grande ampleur des concepts présentés au chapitre 2. Il s'agit de modéliser diagrammatiquement un problème de raisonnement analogique.

Le problème choisi pour modéliser le raisonnement par analogies est la résolution de tests de Q.I. dans sa version la plus simple, ce genre de problème consiste à compléter une série de trois figures. Le raisonnement utilisé lors de cette tâche est analogique et non supervisé (la solution n'est pas choisie parmi une liste de propositions).

Notre objectif est de concevoir et développer un système qui utilise des outils de la topologie algébrique pour résoudre ce type de problème. Notre système, nommé ESQIMO, a été implémenté en `Mathematica`. Nous exposerons sa conception et donnerons de nombreux exemples de réponses produites par ESQIMO.

Notre approche pour résoudre diagrammatiquement ce type de problème consiste à représenter le rapport de deux figures par un chemin dans un certain espace. La création d'une quatrième figure par analogie consiste alors à déformer une figure le long de ce chemin.

Nous commençons par décrire précisément le problème, puis nous esquissons quelques uns des très nombreux travaux qui ont été menés sur l'analogie. Nous présentons ensuite les réponses aux tests de Q.I. fournies par des sujets humains, au cours d'une courte expérience que nous avons organisée. Ces réponses nous permettront d'apprécier la diversité des justifications produites.

Dans la troisième partie du chapitre, nous décrivons précisément l'approche du système ESQIMO, ainsi que l'algorithme utilisé pour produire diagrammatiquement l'analogie. Nous donnons ensuite des exemples commentés de résolutions de tests par ESQIMO.

L'algorithme ESQIMO est ensuite largement discuté. Nous examinons comment il pourrait être étendu, ainsi que les hypothèses sur lesquelles il repose. Enfin, plusieurs généralisations sont proposées.

4.1 Les problèmes du type test de Q.I.

4.1.1 Définition et présentation

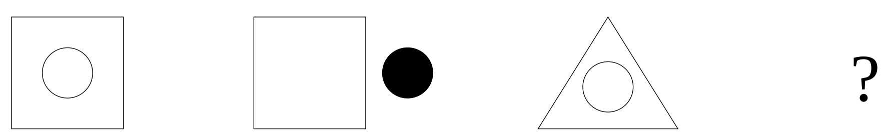
Le problème que nous nous proposons d´étudier revient plus généralement à résoudre une analogie à quatre membres du type :

A est à B, ce que C est à D

où A, B et C sont donnés et peuvent être composés d'éléments d'un univers qui reste à définir. Les figures 4.1 et 4.2 montre deux énoncés typiques de ce genre de test, l'un basé sur des éléments de nature numérique, l'autre basé sur des éléments de nature géométrique.

(a) Figure A	(b) Figure B	(c) Figure C	(d) Figure D
6	12	10	?

FIG. **4.1** – Présentation d'un énoncé typique des questions posées dans les tests de Q.I., avec des éléments de nature numérique



(a) Figure A (b) Figure B (c) Figure C (d) Figure D

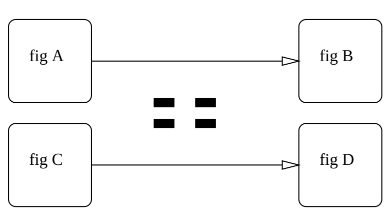
FIG. **4.2** – Présentation d'un énoncé typique des questions posées dans les tests de Q.I., avec des éléments de nature géométrique

Il s'agit, lors de cette tâche, de compléter une suite trois figures par une 4^e de sorte que cette 4^e figure soit à la troisième, ce que la 2^e est à la $1^{\text{ère}}$ (voir figure 4.2). Plus exactement, il est typiquement présenté au sujet un premier couple de figures A et B qui sont affirmées être en rapport non précisé. Ensuite, une troisième figure C est présentée pour être complétée par le sujet en un couple (C, D) qui vérifie le même rapport que le couple (A, B), à déterminer.

Pour la modélisation, nous considérons que le sujet doit obligatoirement passer par au moins une étape intermédiaire où il s'intéresse au rapport entre les figures A et B. Ce rapport est ensuite adapté pour être appliqué à C et ainsi obtenir une solution considérée comme acceptable. Ce qui va nous intéresser ici est justement le jeu de rapports existant entre A, B, C et D.

Nous ne cherchons pas à savoir ce que mesure ce genre de test. De plus, nous ne nous intéresserons pas à une solution unique mais à toutes les solutions engendrables dans la mesure où nous sommes capables de les justifier.

Le rapport analogique à résoudre dans ce genre de problèmes, est schématisé par la figure 4.1, où l'on voit que relation entre A et B et analogue à la relation entre C et D.

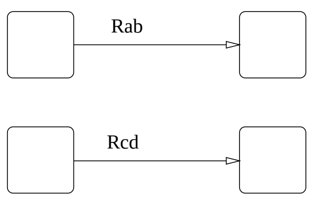


TAB. 4.1 – *Rapport d'analogie dans un test de Q.I géométrique*

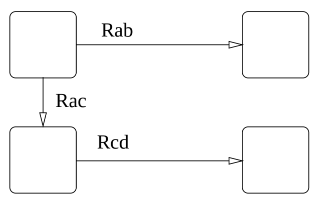
On décompose habituellement la résolution de cette analogie en quatre étapes :

- Première étape : déterminer la(les) relation(s) possible(s) entre A et B : R_{AB}.
- Deuxième étape : déterminer la(les) relation(s) possible(s) entre A et C : R_{AC}.
- Troisième étape : appliquer R_{AB} à C en se restreignant au domaine déterminé grâce à R_{AC}.
- Quatrième étape : vérifier la correspondance entre B et D par la relation R_{AC}.

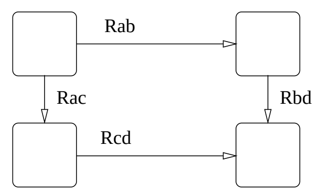
En fait, il existe plusieurs niveaux de résolution de cette analogie. A chaque niveau, on considère une symétrie supplémentaire dans le rapport, ce qui est schématisé par la figure 4.2. Au premier niveau, c'est la relation R_{AB} qui est analogue à la relation R_{CD}. Au deuxième niveau, il faut considérer la relation R_{AC} afin de déterminer le domaine de C sur lequel s'applique l'analogie. Au troisième niveau, enfin, on considère que la relation R_{BD} doit être l'analogue de la relation R_{AC} et le rapport est parfaitement symétrique.



(a) Niveau 1 d'analogie



(b) Niveau 2 d'analogie



(c) Niveau 3 d'analogie

TAB. 4.2 – *Le premier niveau montre une analogie où seules les relations R_{AB} et R_{CD}, sont analogues entre elles. Le deuxième niveau tient compte du rapport R_{AC}, dans la détermination de l'analogie entre les relations R_{AB} et R_{CD}. Le troisième niveau impose une analogie complètement circulaire, les relations R_{AB} et R_{CD} sont analogues, et les relations R_{AC} et R_{BD} aussi.*

4.1.2 Théories sur l'analogie

L'analogie, pour Aristote, est présentée par la métaphore proportionnelle mathématique. Selon cette métaphore, si une paire (a, b) réalise un rapport, une deuxième paire (c, d) lui sera *analogue* si le rapport de cette dernière est le même :

$$\frac{a}{b} = \frac{c}{d}$$

De cette notion mathématique et précise d'analogie par égalité des rapports (la proportion est préservée entre les deux couples), on passe par la suite, à une notion de similitude entre groupes de termes. En particulier, les termes de l'analogie ne sont plus obligatoirement de même nature, on notera alors le rapport d'analogie :

$$a : b :: c : d$$

Cette similitude peut être envisagée d'au moins deux points de vue [Yvo96] :

- synonyme de ressemblance et
- analogie de structure.

Nous pouvons reprendre ces deux points de vue en termes de similitude de surfaces et similitude de structures. Dans la deuxième acception du terme, nous nous attarderons sur la principale théorie qui est due à Gentner (voir le livre de M.-D. Gineste [Gin97] pour une introduction). Gentner inscrit sa théorie dans la conception générale des modèles propositionnels de la représentation des connaissances dans la mémoire humaine. Celles-ci sont considérées comme représentées et représentables, sous la forme de groupes structurés de propositions. Ces propositions formées de prédicats et d'arguments ressemblent, en quelque sorte à des poupées russes, en ce qu'elles incluent des unités elles-même décomposables en propositions plus simples [Gin97]. Ainsi, un argument peut être un concept, pris comme totalité, ou bien un composant de ce dernier.

De façon générale, Gentner retient l'idée que des connaissances analogues sont caractérisées par une identité de la structure des relations qui unissent leurs objets respectifs. Chaque domaine analogue à un autre peut ainsi servir de modèle et permet d'utiliser une structure connue pour en traiter une moins connue [Gin97].

Afin d'extraire cette similitude structurelle ou même fonctionnelle, il faut passer par l'identification d'un rapport de proportion. Pour cela, Gentner introduit une opération de projection, régie par trois règles fondamentales :

- négliger les attributs des objets ;
- préserver les relations entre les objets ;
- choisir des systèmes de relations qui se correspondent au niveau d'ordre le plus élevé.

D'autres travaux dûs à Holyoak sont à citer [Gin97]. Ce dernier considère que ce n'est pas le nombre de ressemblances mais leur nature qui est déterminante. Il propose sa théorie du Schéma, dans laquelle il explique que la transformation d'une situation source vers une cible est une résolution de problème.

Enfin, de multiples travaux portent sur l'efficacité de l'apprentissage grâce aux transferts analogiques, notamment en pédagogie. L'étude des phénomènes liés à l'analogie est très développée en Psychologie et les applications se situent surtout dans le domaine de la Linguistique.

4.1.3 Programmes existants pour la résolution des tests de Q.I.

Le domaine de la résolution de problèmes de type tests de Q.I. a déjà fait l'objet de plusieurs études en Intelligence Artificielle. Nous présentons ici l'approche des deux principaux simulateurs développés en IA. Le premier, le système ANALOGY, a été proposé par Evans en 1968 [Eva68], le deuxième, ARCS et son cousin ACME a été présenté en 1990, par Thagard, Holyoak et *al* [THNG90].

Le système proposé par Holyoak et Thagard est en accord avec le modèle d'extraction et de catégorisation de Holland que nous avons présenté dans le chapitre précédent.

Le système ANALOGY

Le système ANALOGY [Ha91] est destiné à résoudre des tests d'intelligence fondés sur des analogies géométriques et graphiques. Il fonctionne à partir de la connaissance de :

- relations topologiques ;
- relations et transformations géométriques ;
- modifications (suppression ou addition de tracé).

La tâche consiste à sélectionner la bonne figure parmi un choix de 5 figures de sorte qu'elle complète correctement l'analogie. La recherche des solutions s'effectue suivant la démarche suivante :

- génération de règles de transformation possibles entre A et B ;
- génération de règles de transformation permettant de passer de C à chacune des figures proposées ;
- comparaison des deux jeux de règles et sélection de la meilleure solution.

Plusieurs règles de transformation peuvent être associées à un couple de figures. De plus, la dernière étape de comparaison implique une certaine *mesure de similitude.* Enfin, remarquons qu'il se peut que le système ne trouve pas de solution adéquate.

Les auteurs insistent sur le fait qu'il y a un choix à effectuer parmi un certain nombre de solutions proposés. En fait, il serait impossible de laisser le sujet construire complètement une solution dans ce genre de tests car il serait bien difficile de juger de l'exactitude de la solution.

Nous avons un avis différent sur ce point puisque nous pensons qu'il est plus intéressant de construire une ou plusieurs solutions à condition que nous puissions les justifier.

Les systèmes ARCS et ACME

ARCS est l'acronyme pour Analog Retrieval by Constraint Satisfaction [THNG90] et ACME est un système parent, développé par les mêmes auteurs.

Ce modèle, proposé par les collègues de Holland, utilise un ensemble de contraintes sémantiques, structurelles et pragmatiques, et décompose le problème en 4 étapes :

- trouver une source plausible pour l'analogie ;
- la mettre en correspondance avec la cible ;
- transférer l'information de la source vers la cible ;
- apprendre l'analogie.

Leur modèle, comme le nôtre, n'a pas la prétention ni l'ambition d'être plus performant qu'un humain mais seulement de modéliser un processus cognitif plausible. Deplus, ils s'intéressent à ces problèmes par le biais de l'*apprentissage* et de la recherche d'information en mémoire en général.

Leur idée directrice est d'appliquer des contraintes en parallèle, ce qui nous séduit beaucoup. En effet, ils cherchent à modéliser le processus d'*insight.* C'est un point qui nous semble tout à fait intéressant à étudier pour une approche cognitive de l'analogie.

Enfin, ce modèle utilise des stratégies de compétition entre les solutions ce qui ne nous concerne pas pour le moment car nous ne sommes pas intéressés par une classification des solutions.

Pour finir, la technique d'implantation choisie par les auteurs est celle des réseaux de neurones ce qui permet bien un traitement distribué et en parallèle. Mais nous avons vu que les CS nous permettaient également cette particularité tout en promettant d'être bien plus adaptés aux traitements spatiaux.

4.2 Étude expérimentale

Avant de nous lancer dans l'implémentation de la simulation d'un processus cognitif, nous devons procéder à une expérimentation informelle.

L'objectif n'est pas d'analyser l'aptitude des humains à résoudre ce genre de tests, mais simplement d'illustrer la diversité des réponses qui peuvent être produites quand le test n'est pas supervisé (c'est-à-dire, lorsque nous ne proposons pas de choisir la solution dans une liste prédéfinie, mais qu'il faut la construire soi-même).

4.2.1 Le protocole

Nous avons effectué un test sur un échantillon non représentatif de 10 personnes, tous des informaticiens de sexe masculin entre 20 et 34 ans. Cependant, le but n'est pas d'en tirer une analyse de la population testée ni même des conclusions quand à ce que ce test est susceptible de mesurer. Nous cherchons uniquement à constituer une base d'exemples des nombreuses solutions qu'un humain peut produire, afin de posséder un matériel varié pour analyser les types de démarches différentes dans le raisonnement diagrammatique.

Le test se compose de 4 devinettes graphiques du type test de Q.I, une illustration de cette série de tests est présentée sur la figure 4.3. Sur un si petit échantillon, les réponses sont déjà variées et nous nous sommes intéressés, dans un deuxième temps, aux justifications qui accompagnaient ces réponses.

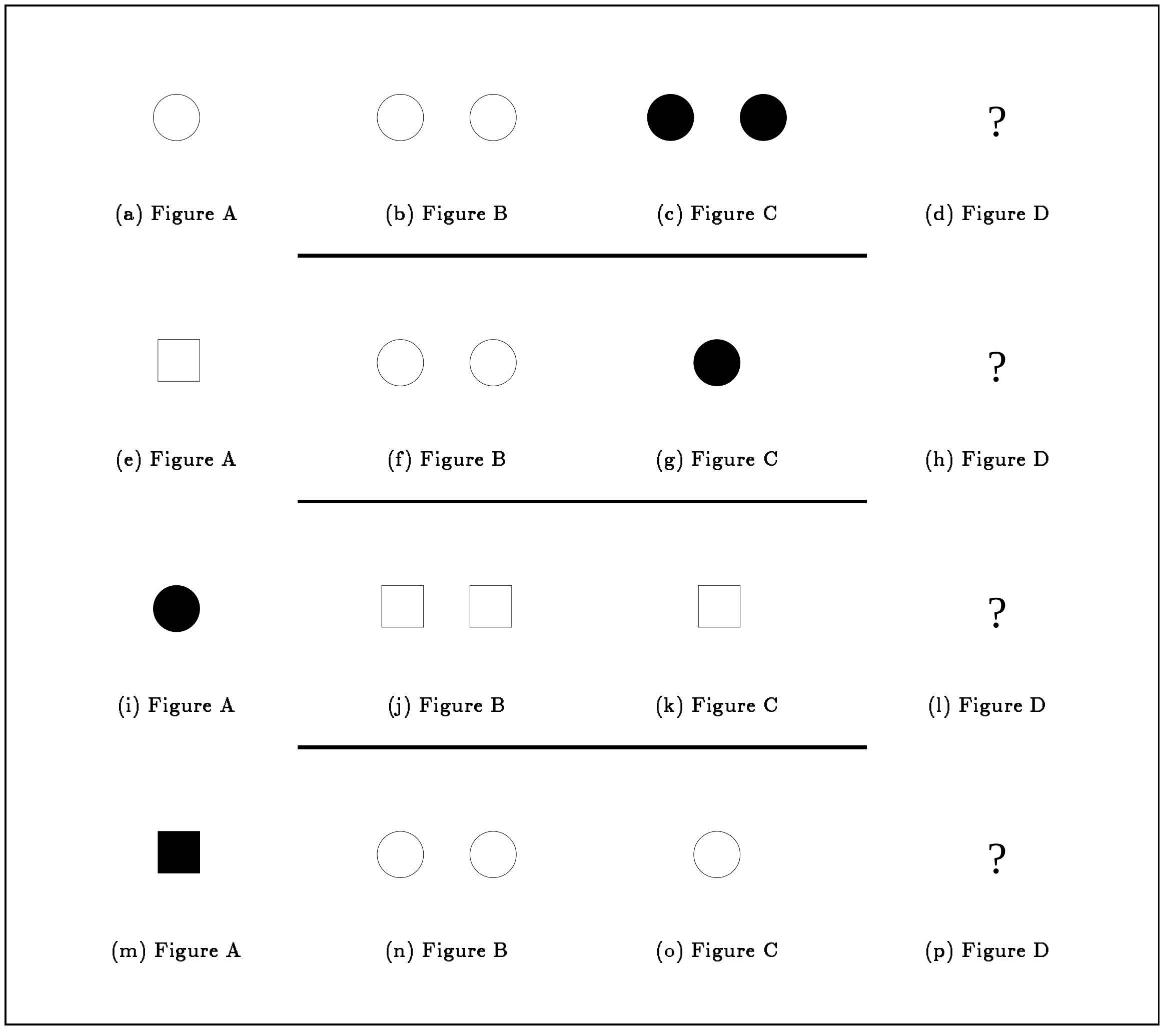


FIG. **4.3** – Les séries A, B, C doivent être complétées par une figure D, de sorte que les 4 figures vérifient le rapport $A : B :: C : D$. La figure D doit être construite, aucun choix de figures n'est proposé.

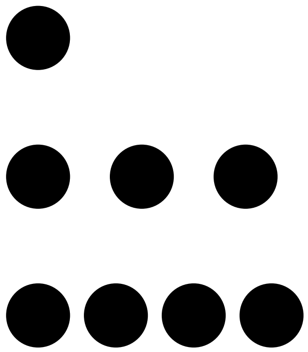
Toute justification et toute réponse est à considérer car elle représente une transformation possible faisant passer de A à B et il s'agit de les envisager toutes.

4.2.2 Résultats et commentaires

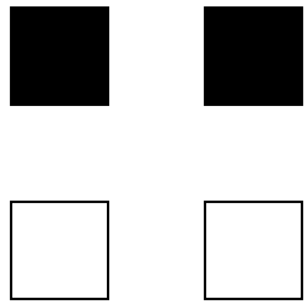
Aucun des tests de la série n'a donné lieu à une réponse unique parmi les 10 sujets. Ceci illustre bien le fait qu'il ne peut en aucun cas exister une seule réponse considérée comme correcte d'après l'énoncé.

Les résultats possibles pour chacune des séries sont donnés par la figure 4.4. Nous avons numéroté chacune des réponses pour porter les fréquences correspondantes sur la table 4.3.

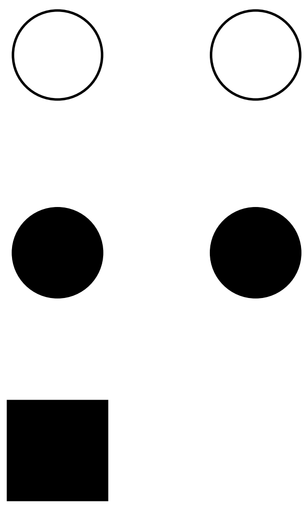
Les différentes solutions données pour un même énoncé ont ensuite été justifiées par les sujets. Voici les justifications fournies pour le premier énoncé, à partir d'une relation supposée exister



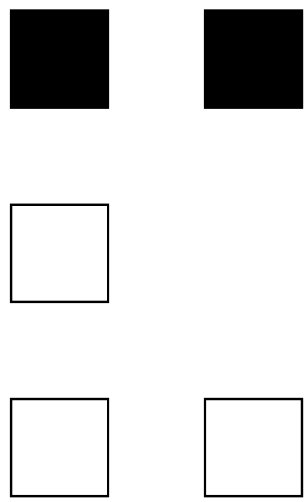
(a) Réponses données au test (1), numérotées de haut en bas $(1a), (1b), (1c)$.



(b) Réponses données au test (2), numérotées de haut en bas $(2a), (2b)$.



(c) Réponses données au test (3), numérotées de haut en bas $(3a), (3b), (3c)$.



(d) Réponses données au test (4), numérotées de haut en bas $(4a), (4b), (4c)$.

FIG. **4.4** – Résultats fournis par les 10 sujets au test de la figure 4.3

entre A et B, appliquée ensuite à C :

- $(1a)$: Le raisonnement porte uniquement sur le nombre de figures et pas sur leur nature. Les sujets disent s'être placés dans un univers binaire, après 2, il y a 1. La relation proposée entre A et B est d'ajouter 1, lorsqu'on ajoute 1 à C, cela fait 3, donc 1, en binaire.
- $(1b)$: Même relation entre A et B, ajouter 1, mais dans un univers décimal (par exemple).
- $(1c)$: La relation entre A et B est maintenant de passer du nombre d'éléments a son double. Ainsi C, contenant 2 figures, en contient 4 de même nature.

Dans cet exercice, les sujets ont raisonné sur le nombre d'éléments puisque la nature des éléments entre les figures A et B ne change pas. Une même solution peut aussi être justifiée de manières différentes. La solution $(1b)$ peut être perçue comme une incrémentation ou comme une multiplication par deux du dernier élément. Ces deux justifications donnent le même résultat ici mais auraient donné un résultat différent si les deux éléments de la figure C avaient été de nature

numéro de réponse	(a)	(b)	(c)
test(1)	4	2	4
test(2)	9	1	-
test(3)	8	1	1
test(4)	8	1	1

TAB. 4.3 – *Réponses fournies à l'expérience des tests de Q. I. de la figure 4.3*

différente. Regardons maintenant les justifications proposées pour le deuxième énoncé :

- $(2a)$: L'univers est encore considéré comme binaire, mais cette fois dans le sens où il n'existe comme figures possibles que des carrés et des ronds. Dans la mesure où l'exemple propose un changement de forme de carré vers rond, il faut changer cette fois la forme de rond vers carré. Cette première relation est combinée à une relation sur le nombre d'éléments, qui passe de un à deux.
- $(2b)$: Étrangement, la couleur des figures est changée dans cette solution. L'argument est que la couleur de la figure d'arrivée est blanche, il faut donc garder cette caractéristique.

Nous voyons à partir de cet énoncé, que les attributs des figures peuvent être considérés comme déterminés ou indéterminés. Par exemple, ceux qui ont répondu $(2a)$, considèrent qu'il n'y a pas lieu de changer la couleur et conservent la couleur *noire*. Ceux qui ont répondu $(2a)$, considèrent que même si la couleur entre A et B n'a pas changé, il faut tout de même s'intéresser à la couleur de la figure d'arrivée et prendre la même pour D. Certains s'intéressent aux attributs de la figure d'arrivée, d'autres aux attributs qui ont changé seulement entre A et B.

Pour le troisième énoncé :

- $(3a)$: Ici la relation considérée, consiste en le changement de forme qui passe de *rond* à *carré* ou de *carré* à *rond* dans un univers à deux seules formes possibles, ainsi que la multiplication par deux du nombre des éléments.
- $(3b)$: La relation est la même que pour la réponse $(3a)$, à ceci près que le changement de couleur est également pris en compte ici.
- $(3c)$: Seul le changement de couleur a été appliqué à C.

Enfin, voici les justifications fournies pour le quatrième énoncé :

- $(4a)$: Changement de couleur, de forme et multiplication par deux.
- $(4b)$: Changement de forme.
- $(4c)$: Changement de forme et multiplication par deux.

Nous pourrions considérer qu'une solution n'est juste que si elle fait intervenir la plus grande relation entre A et B, c'est-à-dire que le fait d'omettre le changement de couleur, ou bien la multiplication par deux dans le cas de la réponse $(4b)$, sont des erreurs. Ici, nous ne tirerons pas ce genre de conclusion, ce sont bien tous ces processus sans exclusion que nous cherchons à modéliser, ils sont tous à inclure dans ce que nous avons appelé le raisonnement diagrammatique.

4.3 Présentation du système ESQIMO

Nous présentons le programme que nous avons implémenté sur la base des remarques précédentes. Dans la suite, le simulateur portera le nom ESQIMO (ESpace-QI-MOdèle).

4.3.1 Modélisation des données du problème

Pour modéliser ces raisonnements, nous allons nous donner des éléments géométriques simples avec des propriétés très simples aussi. Ces éléments seront les seuls connus dans notre univers de départ.

Les éléments géométriques sont choisis pour être les mêmes que ceux présentés dans l'étude de Weber « A testbed for miniature language acquisition » [WS90] car il a les avantages de simplicité tout engendrant une certaine variété nécessaire pour nos commentaires. Comme lui, nous allons choisir un petit nombre de propriétés géométriques de base sans tenir compte des considérations métriques, des orientations dans l'espace ou des distances.

En fait, nous ne devrions pas parler d'éléments en tant qu'individuels. D'après ce que nous avons dit dans la section précédente, ce sont les propriétés que les éléments partagent entre eux qui vont nous intéresser.

Nous choisissons quelques propriétés de forme, de taille et de couleur. Ces propriétés peuvent être représentées à l'aide d'instances connues mais pourront être augmentées d'autre instances si elles sont nouvellement perçues. Les propriétés retenues sont :

- *couleur* : plein ou creux ;
- *taille* : petit ou grand ;
- *formes* : rond, carré ou triangle.

Ceci nous fait une famille de sept propriétés. Remarquons que dans cette famille de propriétés nous ne tenons pas compte de la position relative des éléments dans une figure (au-dessus de, à gauche de, ...).

Bien entendu, ces propriétés ne sont pas fournies au système de manière groupée autour des thèmes de couleur, forme et taille. Les sept propriétés seront traitées de manière indépendante les unes des autres de sorte que nous incluons le minimum de connaissance *a priori* dans notre simulateur. De plus, nous avons choisi trois formes pour que toutes les relations ne soit pas complémentaires deux à deux.

Nous allons tout de même nous donner des éléments représentant l'intersection de ces propriétés. Il pourrait y en avoir plusieurs dans une intersection mais nous en choisissons le minimum. Ces éléments doivent être perçus comme des classes dans la terminologie objet, les instances correspondant aux éléments effectivement présents dans les figures A, B, C et D.

La totalité des éléments manipulés par le système est présentée en figure 4.5. Nous les nommons :

$$\Omega = Univers = \{e_1, e_2, ..., e_{12}\}$$

.

Cela veut dire que les figures A, B, C et D seront composées à partir d'éléments de Ω. Ainsi, si la solution n'est pas supervisée, les éléments de la solution, eux, le sont.

4.3.2 Représentation des propriétés connues dans l'univers Ω

Nous devons construire un ensemble de simplexes représentant les propriétés que partagent les familles d'éléments connus de notre univers, c'est ce que nous appellerons le complexe $C(\Omega)$.

Pour le construire, nous commençons par construire le simplexe de chacunes des relations. Par exemple, la propriété de *rondeur* sera représentée par un simplexe dont tous les sommets seront les représentants connus par notre système comme ayant la propriété de rondeur. Ce simplexe possède 4 sommets e_1, e_4, e_7, et e_{10}, il est donc de dimension 3 :

$$\sigma(\text{rondeur}) = \langle e_1, e_4, e_7, e_{10} \rangle$$

Un 3-simplexe est représenté graphiquement par un tétraèdre plein, ainsi, la propriété *rondeur* est illustrée par la figure 4.6

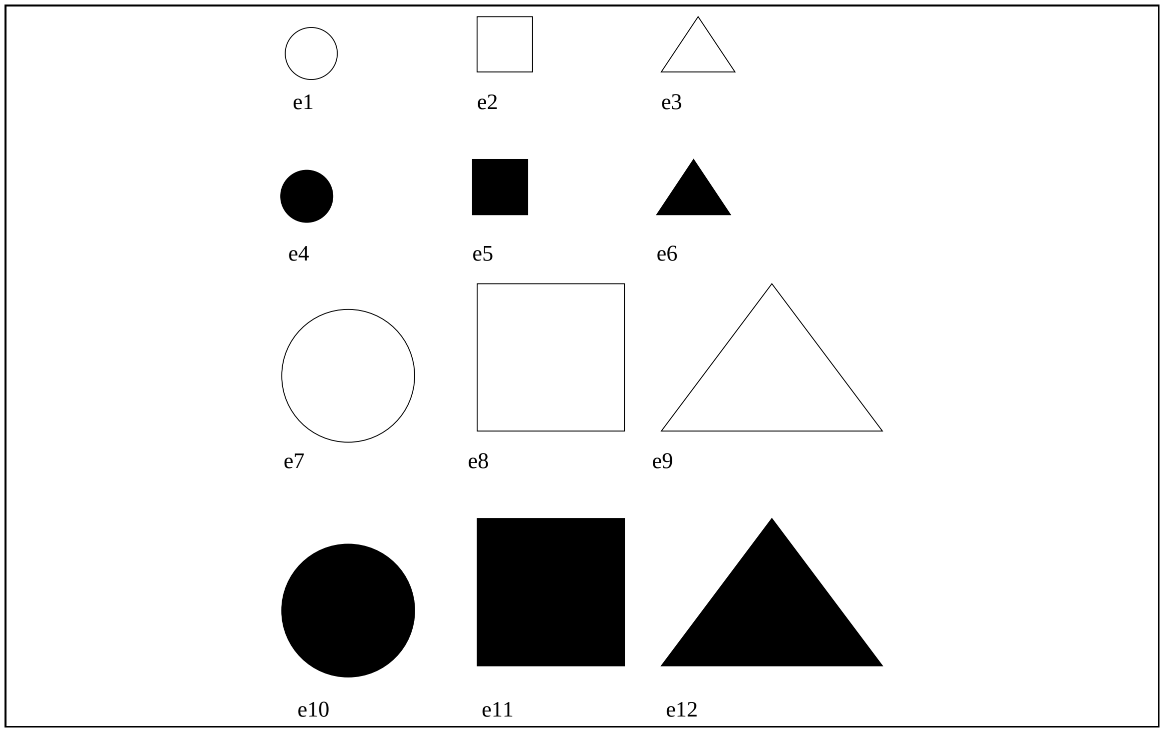
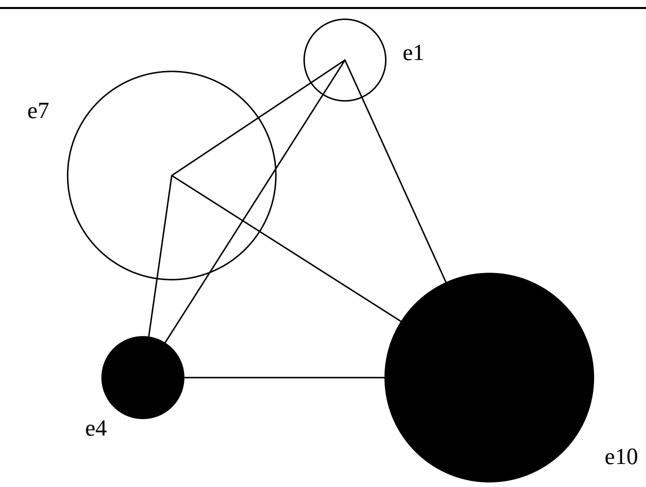


FIG. **4.5** – Élements de base manipulés par le système ESQIMO : c'est le produit cartésien *couleur* $\times$ *taille* $\times$ *forme*, de cardinalité $2 * 2 * 3 = 12$.



Relation de rondeur dans l'univers Ω : Nous avons dessiné la forme géométrique des éléments e_1, e_4, e_7 et e_{10}, pour faciliter la compréhension.

FIG. **4.6** – Relation de rondeur dans l'univers Ω

De même, les relations de *triangularité* ou de *carréité*, seront représentées par des 3-simplexes selon la même technique :

$$\sigma(\text{carréité}) = \langle e_2, e_5, e_8, e_{11} \rangle$$

$$\sigma(\text{triangularité}) = \langle e_3, e_6, e_9, e_{12} \rangle$$

Par contre, les simplexes représentant les autres relations auront 6 sommets chacun et seront donc des 5-simplexes :

$$\sigma(\text{plein}) = \langle e_7, e_8, e_9, e_{10}, e_{11}, e_{12} \rangle$$

$$\sigma(\text{creux}) = \langle e_1, e_2, e_3, e_4, e_5, e_6 \rangle$$

$$\sigma(\text{petit}) = \langle e_1, e_2, e_3, e_7, e_8, e_9 \rangle$$

$$\sigma(\text{grand}) = \langle e_4, e_5, e_6, e_{10}, e_{11}, e_{12} \rangle$$

Il est difficile de représenter ces simplexes autrement que grâce à la représentation en cercle, introduite au chapitre 2. A titre d'exemple, nous avons représenté de cette façon, la propriété *petit* sur la figure 4.7.

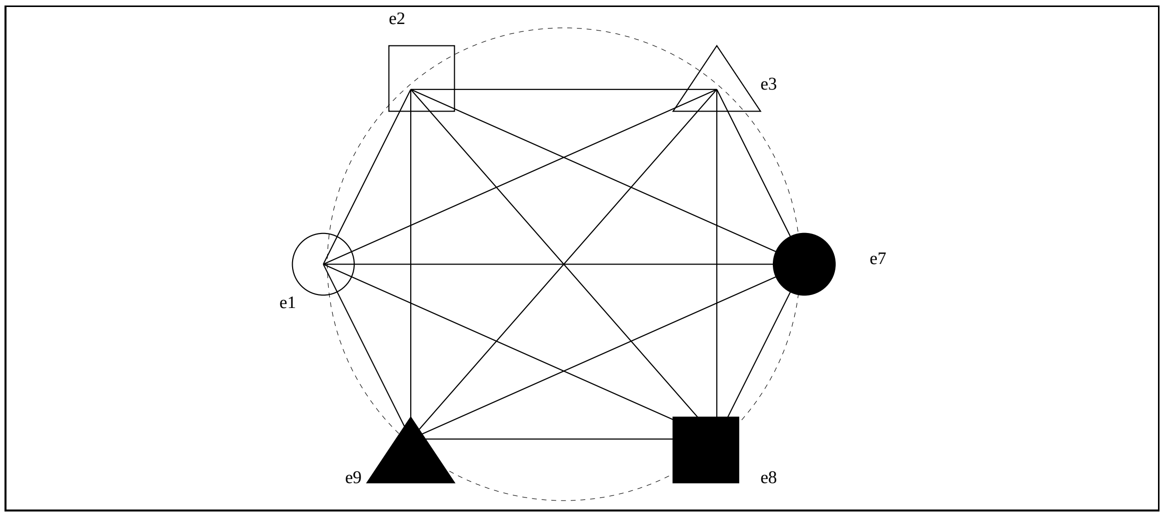


FIG. **4.7** – Représentation polygonale du simplexe *petit* : Nous avons dessiné la forme géométrique des éléments e_1, e_2, e_3, e_7, e_8 et e_9, pour faciliter la compréhension. Les sommets sont les prototypes d'exemples connus de formes partageant la propriété de petitesse, ce ne sont pas des individus. Attention : ceci nést pas un graphe plan mais un polyèdre de dimension 6.

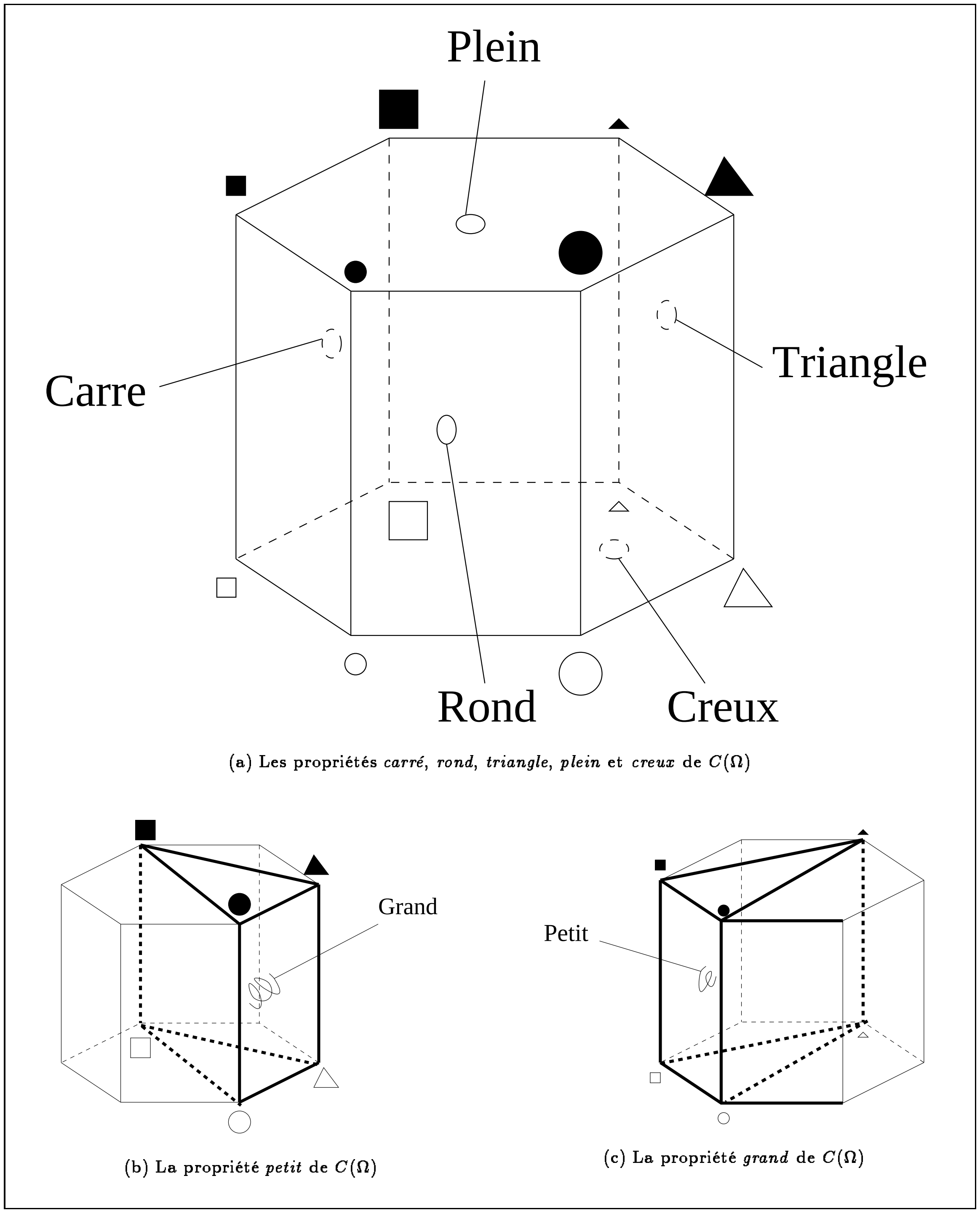
Comme nous l'avons remarqué précédemment, nous pouvons représenter toutes ces propriétés sur un seul et même complexe, en utilisant des dimensions supplémentaires pour chaque relation différente. Ainsi, les éléments de l'univers sont représentés tels qu'ils doivent figurer, pris dans toutes les structures auxquelles ils participent. Nous représentons donc l'état des connaissances sur les éléments de l'univers Ω, par un grand complexe simplicial, nous avons schématisé ce complexe par la figure 4.8, pour en donner une idée intuitive.

4.3.3 Transformation d'un élément en un autre

Nous nous intéressons, tout d'abord, au problème de représenter diagrammatiquement la transformation d'un élément de type e_i en e_j.

Comme le rappelle la figure 4.9, un élément est un simplexe, et ce simplexe appartient à $C(\Omega)$. Ainsi, trouver une transformation de e_i à e_j, correspond à trouver un chemin dans $C(\Omega)$ du simplexe associé à e_i au simplexe associé à e_j.

C'est cette déformation, qui constitue la première étape de la résolution de la devinette graphique par raisonnement diagrammatique, elle est schématisée par la figure 4.10.



(a) Les propriétés *carré*, *rond*, *triangle*, *plein* et *creux* de $C(\Omega)$

(b) La propriété *petit* de $C(\Omega)$

(c) La propriété *grand* de $C(\Omega)$

FIG. **4.8** – Une représentation de $C(\Omega)$. Attention, les simplexes représentant *plein* et *creux* ne sont pas des graphes plans mais des simplexes de dimension 6.

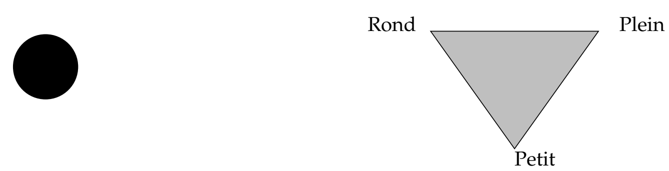
Cette transformation peut ne pas exister, s'il n'existe pas de moyen de passer d'un sommet de A à un sommet de B en passant par des arêtes de Ω. Pour déterminer cela, il faut étudier les q-connections des complexes de A et de B.

Étudions la signification intuitive d'un chemin. Pour cela, considérons deux simplexes σ_1 et σ_2 qui partagent un simplexe σ. Le simplexe σ représente ce qui est resté invariant quand σ_1 se transforme en σ_2. Plus la dimension de σ est grande, plus la similitude entre σ_1 et σ_2 est grande.

Si σ_1 et σ_2 ne sont pas directement connectés par σ, mais qu'ils sont q-connectés par un chemin

	Rond	Carré	Triangle	Plein	Creux	Grand	Petit
e_1	1	0	0	0	1	0	1
e_2	0	1	0	0	1	0	1
e_3	0	0	1	0	1	0	1
e_4	1	0	0	1	0	0	1
e_5	0	1	0	1	0	0	1
e_6	0	0	1	1	0	0	1
e_7	1	0	0	0	1	1	0
e_8	0	1	0	0	1	1	0
e_9	0	0	1	0	1	1	0
e_{10}	1	0	0	1	0	1	0
e_{11}	0	1	0	1	0	1	0
e_{12}	0	0	1	1	0	1	0

TAB. 4.4 – *Matrice d'incidence des e_i par rapport aux propriétés connues dans Ω*



(a) Élément de type e_4

(b) Représentation de cet élément sous forme de simplexe

FIG. **4.9** – Un exemple de représentation sous forme de simplexe d'un élément de type e_4

de longueur p, alors il y a p étapes de transformations élémentaires, chacune d'elles laissant au moins q propriétés invariantes. Évidemment, plus le chemin est long, plus le simplexe de départ et d'arrivée peuvent être différents (même si la dimension du chemin est grande). Nous sommes donc amenés à privilégier naturellement les *chemins les plus courts et de plus grande dimension possible.*

4.3.4 Appliquer une transformation à un complexe quelconque

L'étape suivante consiste à appliquer la transformation qui fait passer d'un simplexe e_i à e_j à un simplexe e_k, et plus généralement à un complexe K.

En effet, il s'agit de trouver les propriétés de la figure C que l'on souhaite transformer selon R_{AB}. Comme nous l'avons vu lors de l'étude expérimentale, certains sujets choisissent de changer tout ce qui est possible en C, d'autres gardent le maximum de caractéristiques de C. Ainsi le domaine de C à transformer varie, et produit des solutions différentes.

Le problème est donc d'étendre une fonction f qui transforme un simplexe donné en un autre simplexe donné, en une fonction $\overline{f}$, de domaine de définition K tout entier.

nous voulons que $\overline{f}$ soit une application simpliciale, c'est-à-dire que si $\sigma_1 \subset \sigma_2$, alors $\overline{f}(\sigma_1) \subset \overline{f}(\sigma_2)$. Cette condition indique simplement que $\overline{f}$ respecte la structure du complexe.

Il y a plusieurs prolongements de f en $\overline{f}$ possibles, chacun correspondant à un raisonnement différent. ESQIMO envisage 4 prolongements différents, qui seront détaillés plus bas.

Nous avons posé les mécanismes de base permettant de résoudre notre problème d'analogie de manière diagrammatique, dans le cas où les figures A et B sont réduites à des simplexes. Ce mécanisme élémentaire de résolution est illustré par la figure 4.11 dans le cas où C est aussi réduit à un simplexe.

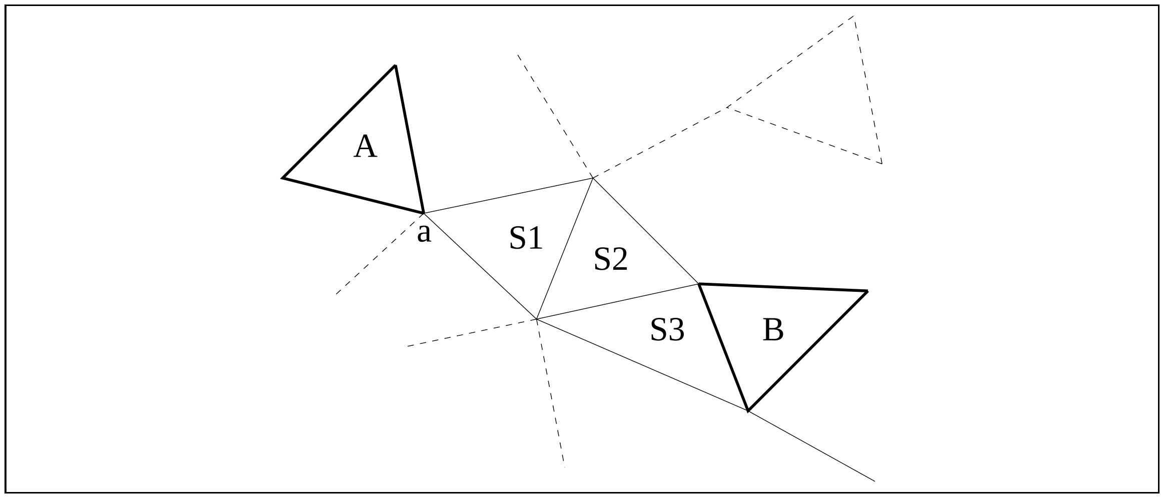


FIG. **4.10** — Le simplexe A se déforme en le simplexe B, le long d'un chemin $(A, S1, S2, S3, B)$, de dimension 0 et de longueur 5. En effet, l'intersection entre A et $S1$ est réduite au 0-simplexe $\langle a \rangle$. Quand A se transforme en $S1$, seule la propriété a est préservée. Mais quand $S3$ se transforme en B, c'est un 1-simplexe qui est préservé.

Cette procédure se généralise immédiatement au cas où C est un complexe, puisque $\overline{f}$ est simpliciale; il suffit d'appliquer *simultanément* $\overline{f}$ à chacun des simplexes de C :

$$\overline{f}(C) = \langle \overline{f}(\sigma_i), \sigma_i \in C \rangle$$

4.3.5 Procédure de résolution générale

Généralisation aux figures A et B non élémentaires

Les figures A, B, et C peuvent être composées d'éléments de type e_i, elles sont alors représentées par des complexes simpliciaux. Ces complexes sont construits à partir des simplexes représentant leur composants, comme le montre la figure 4.12. Nous ne nous intéressons pas à l'orientation de la figure, ni à la distance ou aux positions relatives de ses éléments.

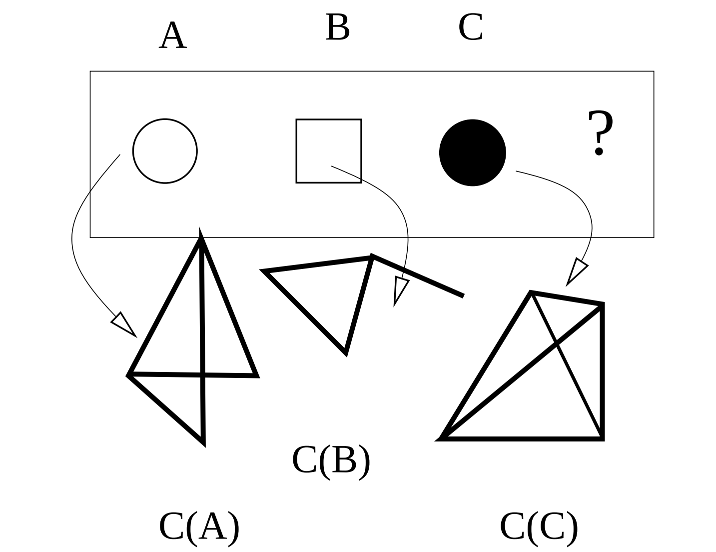
Nous pouvons alors nous ramener au cas précédent, en considérant des appariements des simplexes de A et de B, et l'application de la transformation correspondante, aux simplexes de C.

Appariements des sommets de A et B

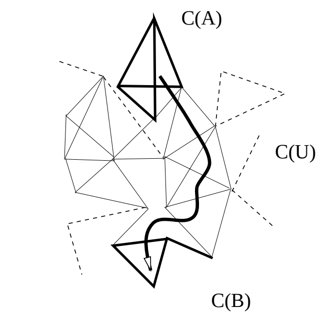
Cet appariement peut être effectué selon plusieurs méthodes. Soit S_a, l'ensemble des sommets de A, et S_b l'ensemble des sommets de B. Si $Card(S_a) = Card(S_b)$, les appariement possibles sont tous ceux qui permettent un recouvrement total de S_b. Par exemple, nous pouvons tout simplement apparier un sommet de A avec un et un seul sommet de B. Toutes les permutations de cet appariement seront aussi acceptables. Mais nous pouvons aussi projeter toute partie de A sur un sommet de B, du moment que tous les sommets de S_b sont appariés. Des exemples d'appariements autorisés quand $Card(S_a) = Card(S_b)$ sont donnés par la figure 4.13

Si $Card(S_a) > Card(S_b)$, il faut que certains sommets de A disparaissent ou bien que plusieurs sommets de A se transforment en un seul sommet de B. Des exemples d'appariements autorisés lorsque $Card(S_a) > Card(S_b)$ sont donnés en figure 4.14. Enfin, si $Card(S_a) < Card(S_b)$, il faut qu'un sommet de A se transforme en plusieurs sommets de B, ou bien qu'il y ait création de sommets, ou que plusieurs sommets de A se transforment en encore plus de sommets dans B. Toutes ces heuristiques d'appariement ont été envisagées et programmées. Des exemples d'appariements autorisés lorsque $Card(S_a) < Card(S_b)$ sont donnés en figure 4.15.

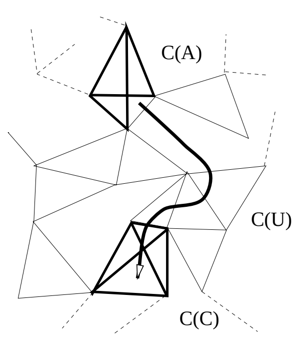
Le fait d'envisager des processus d'appariements où un sommet de A peut être apparié à plusieurs sommets de B, correspond à une déformation simultanée de ce sommet en deux sommets distincts de B. La déformation de A vers B peut se faire selon plusieurs chemins plus ou moins



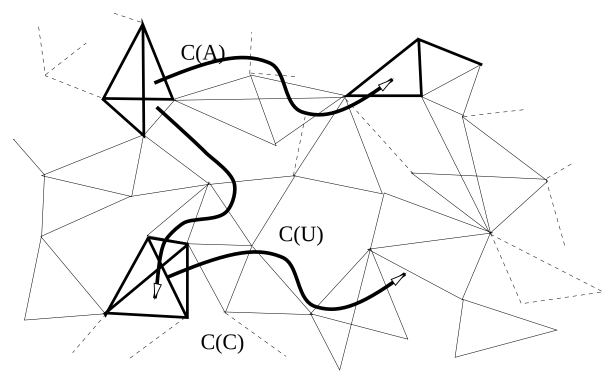
(a) Représentation des figures A, B, C et D sous forme de complexes simpliciaux



(b) Transformation T_{AB} comme cheminement permis dans Ω et rejoignant $C(A)$ avec $C(B)$



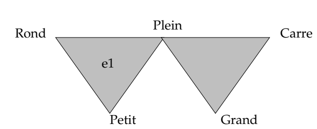
(c) Domaine de C à transformer selon R_{AB}, grâce à R_{AC}



(d) Application de R_{AB} au domaine de C trouvé grâce à R_{AC}, et construction de D

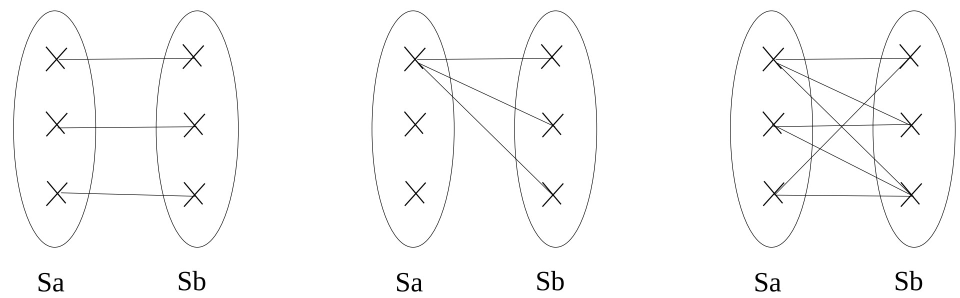
FIG. **4.11** – Les quatre étapes de la modélisation du raisonnement diagrammatique pour la résolution des tests de Q.I, dans le cas où A, B et C sont des simplexes.

(a) Une figure A donnée à représenter



(b) Représentation de la figure A, sous forme de complexe

FIG. **4.12** – Un exemple de figure A pour le problème des devinettes graphiques et sa représentation sous forme de complexe avec la relation de voisinage de base



(a) Appariements 1 − 1, chaque sommet de S_a se transforme en un et un seul sommet de S_b

(b) Tous les sommets de S_a ne sont pas appariés, ceux-là disparaissent

(c) Certains sommets de S_a sont appariés au même sommet de S_b, leurs transformations se confondent.

FIG. **4.13** – Appariements autorisés lorsque $Card(S_a) = Card(S_b)$

(a) Certains sommets de S_a sont appariés sur le même sommet de S_b

(b) Tous les sommets de S_a en trop, sont appariés au dernier sommet de S_b

FIG. **4.14** – Appariements autorisés lorsque $Card(S_a) > Card(S_b)$

(a) Certains sommets de S_a sont appariés sur le même sommet de S_b

(b) Le dernier sommet de S_a est apparié avec tous les sommets de S_b en trop

FIG. **4.15** – Appariements autorisés lorsque $Card(S_a) < Card(S_b)$

longs de $C(\Omega)$. Cela donne bien lieu à différentes solutions, ou bien à plusieurs justifications d'une même solution.

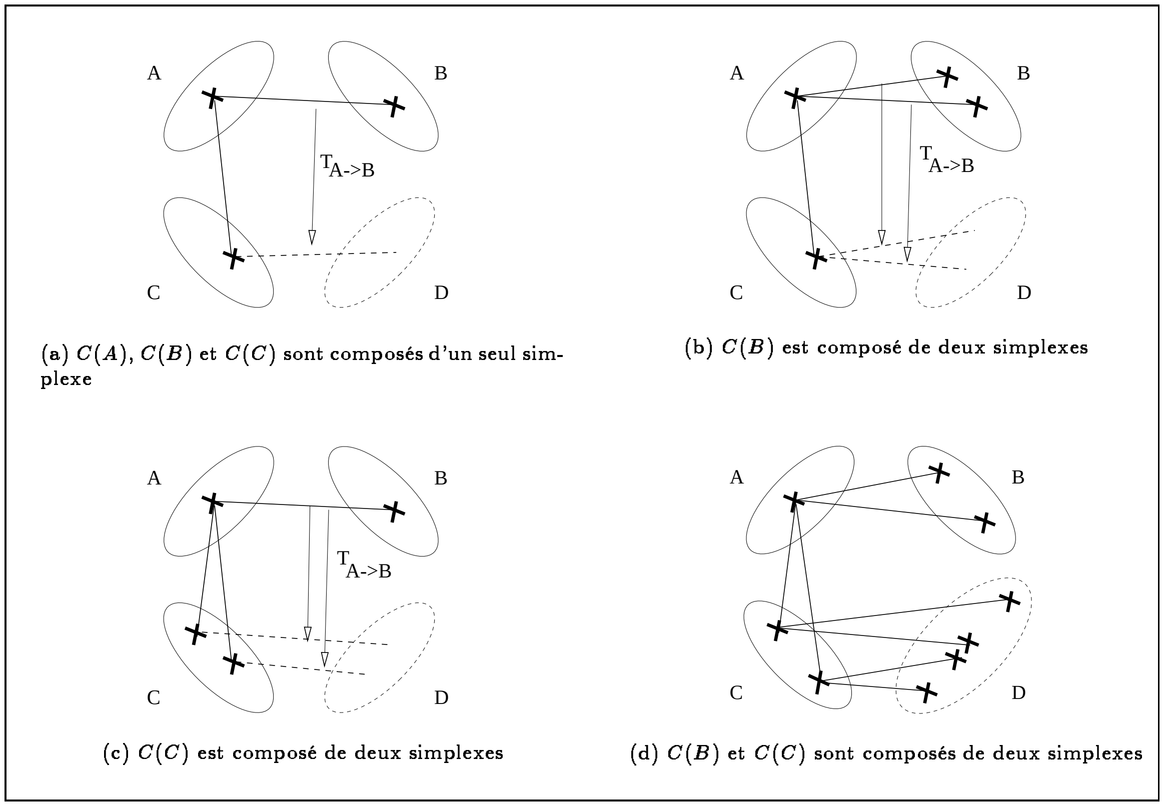
Pour chaque appariement établi, nous déterminons une déformation élémentaire associée pour passer de chacun des éléments de la paire appariée à l'autre. Ce sont ces déformations élémentaires que nous appliquerons ensuite en parallèle sur un domaine de C qu'il reste encore à déterminer. Les déformations élémentaires trouvent une réalisation soit successive soit simultanée selon la déformation considérée.

Application de la transformation à C, construction de la solution D

C'est par le même processus d'appariements que nous nous proposons de trouver le domaine de C, sur lequel appliquer la transformation trouvée pour passer de A à B.

Pour appliquer la transformation R_{AB} sur le domaine de C que nous venons de déterminer, il suffit de déformer le domaine de C, le long de la déformation $A \to B$. Les déformations peuvent être nombreuses, les solutions seront alors nombreuses. Plusieurs déformations peuvent aboutir à une même solution, cela correspond à des justifications différentes données par des sujets à une même réponse.

Un appariement des sommets des représentations de A et de B doit donc être mis en correspondance avec un appariement des sommets de $C(A)$ avec les sommets de $C(B)$. Dans le cas où A, B et C contiennent un seul simplexe, disons σ_A, σ_B et σ_C respectivement, il suffit d'appliquer la transformation $(\sigma_a \to \sigma_B)$ à σ_C, comme le montre la figure 4.16(a).



(a) $C(A)$, $C(B)$ et $C(C)$ sont composés d'un seul simplexe

(b) $C(B)$ est composé de deux simplexes

(c) $C(C)$ est composé de deux simplexes

(d) $C(B)$ et $C(C)$ sont composés de deux simplexes

FIG. **4.16** – Application de R_{AB} sur les simplexes de $C(C)$, et construction de $C(D)$. Les croix symbolisent les simplexes contenus dans la représentation des figures, les ensembles symbolisent les complexes associés aux figures A, B, C et D. Les flèches en pointillés correspondent à l'application de R_{AB} sur les composantes de $C(C)$. Dans les cas 4.16(b) et 4.16(d), l'application des différents appariements mène à de nombreuses solutions possibles pour D.

Dans le cas où B contient deux simplexes, σ_B et σ'_B, on dispose de deux transformations $(\sigma_A \to \sigma_B)$ et $(\sigma_A \to \sigma'_B$ que l'on applique alors en *parallèle* sur σ_C (voir figure 4.16(b)).

Dans le cas où C contient deux simplexes σ_C et σ'_C, il y a plusieurs appariements envisageables entre A et C. Si σ_A et apparié à σ_C *et* σ'_C, alors $(\sigma_A \to \sigma_B)$ est appliqué en parallèle à σ_C et σ'_C (voir figure 4.16(c)).

Enfin, dans le cas général, notons σ^a_B, les simplexes de B, liés au simplexe a de A, et σ^a_C, les simplexes de C liés à a. Alors, les transformations $(a \to \sigma^a_B)$ sont à appliquer aux simplexes σ^a_C (voir figure 4.16(d)).

4.3.6 Implémentation commentée

Dans cette implémentation, nous utiliserons une représentation alternative des complexes abstraits sous forme d'ensemble de lignes de la matrice d'incidence de $C(\Omega)$. Un simplexe est donc un vecteur, et un complexe, un tableau. Cette représentation est possible, puisque nous travaillons toujours dans $C(\Omega)$, qui est donné *a priori*. Elle n'est pas utilisable quand les sommets de l'univers ne sont pas connus, ou s'ils peuvent évoluer dynamiquement. Cette implémentation des complexes a l'avantage d'être bien adaptée aux traitements que nous avons à effectuer dans ESQIMO.

Définition et fonctions de base

Nous commençons naturellement par définir les éléments de l'univers Ω, appelé ici `Univers`.

```
Univers = { e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12};
```

Les relations sur Ω, sont appelées `PleinQ`, `CreuxQ`, `RondQ`, `CarreQ`, `triangleQ`, `GrandQ`, `PetitQ`, et définies pour chacun des éléments, par exemple voici `PleinQ` :

```
PleinQ[e4]:= True;
PleinQ[e5] := True;
PleinQ[e6]:= True;
PleinQ[e10] := True;
PleinQ[e11]:= True;
PleinQ[e12] := True;
PleinQ[_] :=False;
```

Pour représenter les éléments de `Univers` par des simplexes, il suffit d'extraire la ligne de la matrice d'incidence pour l'élément, comme nous l'avons montré précédemment. La fonction `LigneDe`, permet cette opération, la matrice totale étant toutes les lignes associées à tous les éléments de l'univers, à savoir `UniversLignes` :

```
LigneDe[e_] := Map[#[e]&,ListeProp]
ExtraitLignesDe[f_cFigure] :=Map[LigneDe,ObjectFrom[f]]
ExtraitLignesDe[u_] :=Map[LigneDe,u]
UniversLignes = ExtraitLignesDe[Univers];
```

Afin de déterminer si deux simplexes sont connectés, nous écrivons une fonction `Qconnection` qui calcule la q-connection. Comme les simplexes sont représentés par leur lignes dans la matrice d'incidence, c'est la fonction `PartageDeLignes`, qui vérifie effectivement si deux simplexes sont connectés.

```
PartageDeLignes[l1_, l2_] :=Count[MapThread[And,{l1,l2}], True]
Qconnection[q_Integer][l_, tab_] :=Select[tab, (PartageDeLignes[l,#] >= q)&]
```

Les appariements de sommets de simplexes

La première étape de l'algorithme présenté est la recherche d'appariements, nous avons programmé deux heuristiques, nommées `App1` et `App2`. La fonction `Appariements` appelle l'une de ces deux heuristiques (avec la deuxième qui est appliquée par défaut).

```
Appariements[l1_List, l2_List, fct_:App2] := fct[l1,l2]
```

La première fonction, `App1`, correspond à l'implémentation de la première heuristique. Tous les appariements de n'importe quelle partie de l'ensemble S_a avec chaque sommet de S_b sont calculés, et la fonction renvoie un ensemble d'appariements désigné par la tête `AssocSet`.

```
App1[l1_List, l2_List]:=Regroupe[l1]/@(App1[l1,l2,{AssocSet[]}])
App1[l1_List,{},r_List]:=r
App1[l1_List,{x_,l___},r_List]:=
    App1[l1,{l},Flatten[UpdateAssocSet[r,x]/@l1,1]]
UpdateAssocSet[r_List, y_][x_]:=Map[Join[#,AssocSet[FromTo[x,y]]]&,r]
Regroupe[l1_List][s_AssocSet]:=
  AssocSet@@Map[FromTo[#,Cases[s,FromTo[#,y_]:>y]]&,l1]
```

La deuxième fonction, App2, correspond à l'implémentation de la deuxième heuristique. Ici, on procède à un appariement $1-1$ dans l'ordre de classement des sommets de A et de B. S'il reste des sommets de S_a non appariés, ils disparaissent, ce qui revient à les apparier à l'ensemble $\emptyset$ (voir figure 4.14(b)). S'il reste des sommets de S_b, non appariés, alors c'est le dernier sommet de S_a apparié, qui est apparié aux sommets de S_b restant (voir figure 4.15(b)).

```
App2[l1_List,l2_List] := {AssocSet@@
            Which [
                (Length[l1] === Length[l2]), MapThread[FromTo[#1,{#2}]&,{l1,l2}],
                (Length[l1] > Length[l2]),
 Join[MapThread[FromTo[#1,{#2}]&,{Take[l1,Length[l2]],l2}],
 FromTo[#,{}]&/@Drop[l1,Length[l2]]]],
                (Length[l1] < Length[l2]),Join[
                    MapThread[FromTo[#1,{#2}]&,{Drop[l1,-1],Take[l2,Length[l1]-1]}],
            {FromTo[Last[l1],Drop[l2,Length[l1]-1] ]}]]}
```

Les chemins associés aux appariements

Les appariements se traduisent en termes de chemins à parcourir pour aller d'un sommet de l'appariement au sommet d'arrivée de l'appariement. La fonction Chemins détermine les chemins existant entre deux simplexes. Les simplexes sont toujours exprimés en termes de lignes de la matrice d'incidence des relations de l'univers.

```
Chemins[depart_List,arrivee_List, permis_List] /;(
 MemberQ[permis,depart]&&MemberQ[permis,arrivee]):=
    With[{q=Max[Nsommets[depart],Nsommets[arrivee]]},
        Qchemins[depart, arrivee, permis][q, 0, {depart}]
    ]
Chemins[depart_Symbol,arrivee_Symbol, permis_List] :=
  Chemins[LigneDe[depart],LigneDe[arrivee],permis]

Qchemins[depart_, arrivee_, permis_] [q_,n_,ens_]:=
    If[MemberQ[ens,arrivee],ChemSet[q,{Piste[arrivee]}],
        If[(n<=Length[permis]),(*
        (With[{
            ch= Qchemins[depart, arrivee, permis][q,n+1,
                Union@@Map[Qconnection[q][#, permis]&,ens]]},
        UpdateChemSet[ens,q,ch]]),
        If[(q <= 0),ChemSet[],
        Qchemins[depart, arrivee, permis][q-1,0,{depart}]]]]
```

Pour deux sommets appariés, il existe beaucoup de chemins possibles, l'ensemble des chemins renvoyé commence par la tête ChemSet. Chaque ChemSet est composé d'une Piste à suivre pour déformer un sommet en l'autre. Une Piste est un fragment élémentaire de chemin, elle correspond donc à un déplacement élémentaire le long d'un chemin.

```
UpdateChemSet[_,_,ChemSet[]]:=ChemSet[]
UpdateChemSet[s_,q_,ChemSet[q_,chems_]]:=
    With[{nchems=Union[Flatten[Outer[UpdatePiste[q],s,chems,1]]]},
        ChemSet[q,nchems]]
UpdateChemSet[_,_,r_ChemSet]:=r

UpdatePiste[q_Integer][x_,Piste[y_,l___]]:=
    If[PartageDeLignes[x,y]≥q,{Piste[x,y,l]},{}]
```

Déformations élémentaires associées aux appariements

Chaque `Piste`, peut se traduire en termes de déformations élémentaires. En effet, lorsque l'on chemine le long d'un `Piste`, cela revient à changer l'une des propriétés du simplexe qui se déforme. Chaque `Piste` correspond donc à une `DefElem`. Il y a plusieurs heuristiques pour déterminer `DefElem`, c'est pourquoi nous en avons implémenté cinq, nommées `DefElem1` à `DefElem5`. La fonction `CurrentDefElem`, appelle l'une des heuristique, nous choisissons `DefElem1` par défaut. `DefElemm[a, b][s]`, correspond à l'application de la déformation de `a` en `b`, à `s`.

```
CurrentDefElem = DefElem1;
DefElem[a_,b_][argu_List] /;VectorQ[argu]:= CurrentDefElem[a,b][argu]
DefElem[a_,b_][argu_List]:= (CurrentDefElem[a,b][#])& /@ argu
```

Dans cette première heuristique, toutes les propriétés qui changent dans le cheminement le long d'une `Piste`, seront à changer sur le sommet de S_c sur lequel s'appliquera la transformation T_{AB}.

```
DefElem1[a_,b_][argu_List] :=
        With[{change = MapThread[Xor,{a,b}]},
            With[{ posChange = Position[change,True]},
                With[{ un2zero= Select[posChange,a[[First[#]]]&],
          zero2un = Select[posChange, !a[[First[#]]]&]},
                ReplacePart[ReplacePart[argu,False,un2zero],True,zero2un]  ]]]
```

Dans cette deuxième heuristique, pour toutes les propriétés le long de la `Piste` qui passent de a à b, nous appliquerons la transformation à S_c en forçant la valeur de cette propriété à devenir b. Si la valeur pour cette propriété est déjà b, elle reste à b.

```
DefElem2[a_,b_][argu_List] := b
```

Cette fois-ci, la valeur de la propriété qui change passe à b, si et seulement si, la propriété se trouvait effectivement aussi dans l'état a avant l'application de la déformation.

```
DefElem3[a_,b_][argu_List] := If[argu===a,b,argu]
```

Cette quatrième heuristique est une version un peu plus sophistiquée des deux précédentes. Les valeurs des propriétés ne seront changées que si elles sont dans l'état de départ correspondant à a, et que de plus, elles font partie des propriétés communes au départ entre C et A. C'est une manière de forcer le domaine de C, sans passer par les appariements.

```
DefElem4[a_,b_][argu_List] :=
        With[{change = MapThread[Xor,{a,b}],
            domaine=MapThread[(!Xor[#1,#2])&,{a,argu}]},
        With[{inter=MapThread[And,{change,domaine}]},
            With[{ posChange = Position[inter,True]},
            With[{ un2zero= Select[posChange,a[[First[#]]]&],
            zero2un = Select[posChange, !a[[First[#]]]&]},
            ReplacePart[ReplacePart[argu,False,un2zero],True,zero2un]  ]]]]
```

Cette version est encore plus forcée, dans le sens où on oblige un élément à n'avoir qu'une seule propriété de forme, de couleur et de taille avec la valeur 1 à la fois, c'est à dire que l'élément sera obligatoirement stable après application de cette déformation. Cette heuristique possède de la connaissance *a priori* sur les propriétés et sera peu utilisée.

```
DefElem5[a_,b_][argu_List]:=
    With[{changeForm=Or@@MapThread[Xor,{Take[a,{1,3}],Take[b,{1,3}]}],
            changeTaille=Or@@MapThread[Xor,{Take[a,{6,7}],Take[b,{6,7}]}],
            changeCouleur=Or@@MapThread[Xor,{Take[a,{4,5}],Take[b,{4,5}]}]},
        Join[
            If[changeForm && (Take[argu,{1,3}] === Take[a,{1,3}]),Take[b,{1,3}],
 Take[argu,{1,3}]],
            If[changeCouleur && (Take[argu,{4,5}] === Take[a,{4,5}]),Take[b,{4,5}],
 Take[argu,{4,5}]],
            If[changeTaille && (Take[argu,{6,7}] === Take[a,{6,7}]),Take[b,{6,7}],
 Take[argu,{6,7}]]
        ]]
```

Application de la déformation globale

Les DefElem sont groupées pour constituer la déformation globale à appliquer sur le domaine de C. Certaines DefElem seront appliquées simultanément, lorsqu'elles concernent des sommets différents qui se transforment en parallèle. D'autres seront appliquées successivement lorsqu'elles correspondent à des déformations le long d'une même piste pour un sommet donné. Ainsi, nous introduisons les têtes Successivement et Simultanement.

```
Deformation[x_,{},_]:=Laisse[x]
Deformation[x_,{y_},permis_List]:=Successivement@@PisteMin[x,y,permis]
Deformation[x_,l_List,permis_List]/;(Length[l]>1):=
  Duplique@@(Deformation[x,{#},permis]&/@l)
```

De plus, on choisira en priorité d'appliquer la déformation qui correspond à la piste de longueur minimale, calculée à l'aide de la fonction PisteMin.

```
PisteMin[l1_, l2_,permis_List] :=
  With[{ch = Chemins[l1,l2,permis]}, First[Sort[ch[[2]] ]]]
```

La fonction Applique se charge d'appliquer effectivement ces transformations sur une ligne correspondant à la représentation d'un sommet.

```
Applique[f_,{}]:={}

Applique[Successivement[f_],argument_List]:=f[argument]
Applique[Successivement[f_,suite__],argument_List]:=
    Applique[Successivement[suite],f[argument]]

Applique[d_Duplique, argument_List]:=
  Join@@((Applique[#,argument]&)/@(List@@d))

Applique[Laisse[_],argument_List]:={}
```

Résolution complète du test de Q.I.

Nous définissons une fonction AppApp qui se charge de sélectionner un appariement des sommets de A et B et un appariement de A avec C, pour proposer une résolution. Nous avons deux heuristiques correspondant aux deux fonctions AppApp1 et AppApp2.

```
AppApp1[l1_List,l2_List]:={Choisir[First[l1],First[l2]]}
AppApp2[l1_List,l2_List]:=Flatten[Outer[Choisir,l1,l2],1]
```

A partir de cette mise en correspondance, la fonction Resolve effectue dans l'ordre les quatre étapes de la résolution, par les appels aux fonctions définies précédemment.

```
Resolve[A_List,B_List,C_List,
          univers_List:UniversLignes,fctApp_:App2,fctAppApp_:AppApp1]:=
    With[{aa=Range[Length[A]],bb=Range[Length[B]],cc=Range[Length[C]]},
    With[{appAB=Appariements[aa,bb,fctApp],appAC=Appariements[aa,cc,fctApp]},
    With[{appABC=fctAppApp[appAB,appAC]},
          Print["appariement A-B : ", appAB];
          Print["appariement A-C : ", appAB];
          Print["appariement des appariements :  ", appABC];
    With[{paireDeList=
            appABC/.Choisir[a_AssocSet,b_AssocSet]:>
                Simultanement@@Transpose[{List@@a,List@@b}]},
    With[{trans=paireDeList /.  {FromTo[idx_,l1_],FromTo[idx_,l2_]}:>
          Domaine[idx,
          Transformation[Deformation[A[[idx]],Map[B[[#]]&,l1],univers]],

          Map[C[[#]]&,l2]]},
          Print["transformation :",trans//Voir];
    With[{allSolutions=Choisir@@(trans/.Domaine[_,f_,a_]:>
          Applique[f,a] /.Simultanement:>Join)},
    With[{solutions=Union[List@@allSolutions]},
    With[{perfsol=Map[Paire[Count[allSolutions,#], #]&,solutions]},
    With[{bestsol=Sort[perfsol,(#1[[1]]<#2[[1]])&]},
          Print["solutions (avec multiplicité) = ",bestsol//Voir];
          Choisir@@((#[[2]])&/@bestsol)
    ]]]]]]]]
```

4.4 Exemples commentés

Nous présentons des exemples d'utilisation des fonctions du programme, pour illustrer les étapes principales d'une résolution avec ESQIMO. Nous présentons, ensuite, une série d'exemples de résolutions de tests de Q.I. par ESQIMO, exemples que nous commentons.

4.4.1 Exemples d'utilisation des fonctions

Pour représenter une figure A du problème, il faut extraire les lignes correspondant aux simplexes des éléments de la figure. Nous créons ici une figure (`Essai1`), dont nous présentons la représentation sous forme de complexe simplicial.

```
Essai1 = CreerFigure [{{e3,{1,1}},{e8,{2,1}},{e6,{3,1}}}];
ExtraitLignesDe[Essai1]
↪ {{"False", "False", "True", "True", "False", "True", "False"},
   {"False", "True", "False", "True", "False", "False", "True"},
   {"False", "False", "True", "False", "True", "True", "False"}}
```

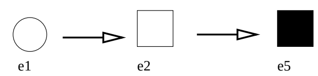
À partir de cette représentation, nous voulons connaître les chemins possibles entre deux simplexes donnés en empruntant les chemins existant dans $C(\Omega)$. Nous rappelons que $C(\Omega)$, est implémenté par la fonction `UniversLignes`.

```
[Chemins[e1,e5,UniversLignes]

↪ { {(Piste[e1, e2, e5])}, {(Piste[e1, e4, e5])} }
```

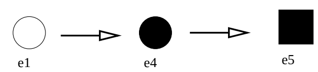
Ainsi, nous voyons que pour passer de e_1 à e_5 dans $C(\Omega)$, nous pouvons passer, soit par e_2, soit par e_4. Cela correspond à passer du cercle, creux, petit, au carré, plein, petit, en passant, soit par un carré, creux, petit, soit par un cercle, plein, petit. En d'autres termes, nous avons les transitions données par la figure 4.17.

Nous pouvons également examiner les appariements proposés par ESQIMO lorsque $C(A)$ et $C(B)$ contiennent chacun deux simplexes, avec la fonction d'appariement par défaut qui est `App2`.

```
Appariements[{e1,e2},{e3,e5}]
```



(a) Première piste proposée en passant par e_2



(b) Deuxième piste proposée en passant par e_4

FIG. **4.17** – Pistes possibles pour cheminer de e_1 vers e_5 dans $C(\Omega)$

```
↪ {AssocSet[FromTo[e1,{e3}],FromTo[e2,{e5}]]}
```

Si la fonction d'appariement utilisée est `App1`, alors, il y a plus d'appariements proposés.

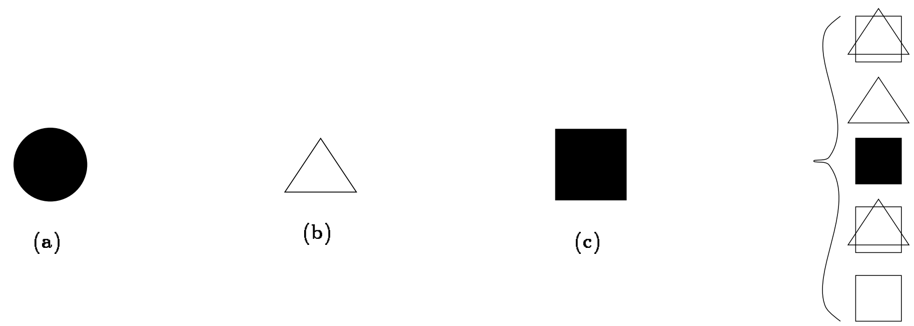
```
Appariements[{e1,e2},{e3,e5},App1] //TableForm
```

```
↪ { {(AssocSet[FromTo[e1, {e3, e5}], FromTo[e2, {}]])}, {(AssocSet[FromTo[e1,
  {e5}], FromTo[e2, {e3}]])}, {(AssocSet[FromTo[e1, {e3}], FromTo[e2, {e5}]])},
  {(AssocSet[FromTo[e1, {}], FromTo[e2, {e3, e5}]])} },
```

Nous comparons maintenant les résultats proposés pour les cinq heuristiques de transformation des `Piste` en `DefElem`. Nous proposons d'appliquer à e_5, la transformation qui fait passer de e_4 à e_3. C'est-à-dire que nous transformons un petit cercle plein, en un petit triangle creux. Puis, nous l'appliquons à un petit carré plein.

```
DefElem1[LigneDe[e4],LigneDe[e3]] [LigneDe[e5]]
DefElem2[LigneDe[e4],LigneDe[e3]] [LigneDe[e5]]
DefElem3[LigneDe[e4],LigneDe[e3]] [LigneDe[e5]]
DefElem4[LigneDe[e4],LigneDe[e3]] [LigneDe[e5]]
DefElem5[LigneDe[e4],LigneDe[e3]] [LigneDe[e5]]
```

```
↪ "Unknown"[False,True,True,True,False,True,False]
  e3
  e5
  "Unknown"[False,True,True,True,False,True,False]
  e2
```



(a) (b) (c)

(d) Cinq solutions posibles correspondant à l'application des cinq heuristiques `DefElem1` à `DefElem5`

FIG. **4.18** – Comparaison des 5 heuristiques `DefElem` de résolution de test de Q.I. avec ESQIMO

La première et la quatrième heuristique produisent des éléments qualifiés d'inconnus (`Unknown`, voir figure 4.18(d)), ils correspondent à des éléments *non-stables*. Nous appelons *stable*, une figure composée des éléments de Ω, et *non-stable*, une figure qui contient au moins un élément jusque-là inconnu dans Ω. Ces deux heuristiques donnent la même réponse, qui est un carré triangulaire petit et creux. Cela correspond à garder la propriété petit, changer la couleur en *creux*, et rendre la figure triangulaire, sans toucher à sa carréitude.

La deuxième heuristique propose transformer e_5 en e_3 (voir deuxième solution de la figure 4.18(d)), comme pour e_4. Cela correspond à tout changer en b, dans une transformation $a \rightarrow b$.

La troisième heuristique, laisse e_5 inchangée, cela correspond à un refus de changer les propriétés de e_5, car il ne correspond pas *exactement* au domaine a, dans une transformation $a \rightarrow b$.

Enfin, la dernière heuristique, transforme e_5 en e_2, c'est-à-dire que la couleur est changée mais pas la forme, car le domaine de définition sur C est restreint aux intersections avec A, or la forme de C ne correspond pas à celle de A.

Examinons maintenant les déformations que le système propose lorsqu'un élément doit être apparié avec deux simplexes. Nous voyons apparaître des déformations *successives* le long d'une `Piste`, et `Duplique`, propose de déformer *simultanément* l'élément de départ en le dupliquant pour en créer deux à l'arrivée.

```
a = Deformation[{e1},{e2,e3},UniversLignes,App2] ;
↪ Choisir[Simultanement[ Duplique[Successivement["D-elem"[RondQ->0,CarreQ->1]],
  Successivement["D-elem"[RondQ->0,TriangleQ->1]]]]]
```

Ici, la déformation consiste à appliquer `simultanement` une seule déformation correspondant à un seul appariement. Cette déformation est obtenue par déformations appliquées `Successivement`, ce qui correspond au cheminement le long d'une `Piste`. Ces déformations successives sont élémentaires, et sont notées `D-elem`.

4.4.2 Exemples de résolution de tests de Q.I.

Regardons maintenant les solutions que ESQIMO propose a une dizaine de tests de QI. Chaque exemple est numéroté, nous en avons fourni l'entrée et la sortie affichée par la système et une illustration géométrique à chaque fois. ESQIMO peut donner la trace de son raisonnement, ce qui constitue une justification de l'analogie. En particulier, il fournit des informations sur :

- les appariements choisis ;

- la transformation qu'il applique ;
- le domaine de C sur lequel il l'applique ;
- l'ensemble des solutions.

Exemple 1

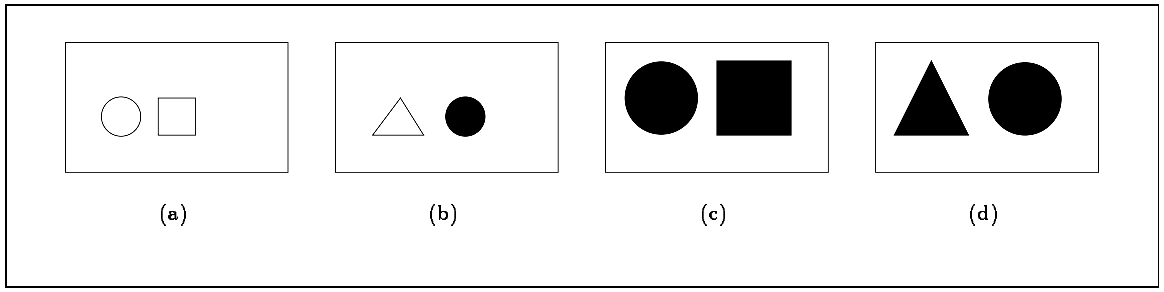


FIG. **4.19** – Exemple 1 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1],LigneDe[e2]};
B1={LigneDe[e3],LigneDe[e4]};
C1={LigneDe[e10],LigneDe[e11]};
Resolve[A1,B1,C1,App2,AppApp2] // Voir
↪ Choisir[AssocSet[FromTo[1,{1}],FromTo[2,{2}]],
  AssocSet[FromTo[1,{1}],FromTo[2,{2}]]]}

  Simultanement[
  Domaine[1,
  Successivement["D-elem"[RondQ->0,TriangleQ->1]],{e10}],
  Domaine[2,
  Successivement["D-elem"[CreuxQ->0,PleinQ->1],
  "D-elem"[CarreQ->0,RondQ->1]],{e11}]]}

  Choisir[{e12,e10}]
```

Justification : La transformation consiste à transformer le rond de gauche en triangle et le carré de droite en rond noir. Ici, la couleur du triangle obtenu à droite est noire car les éléments de départ étaient déjà noirs, contrairement à la figure A.

Exemple 2

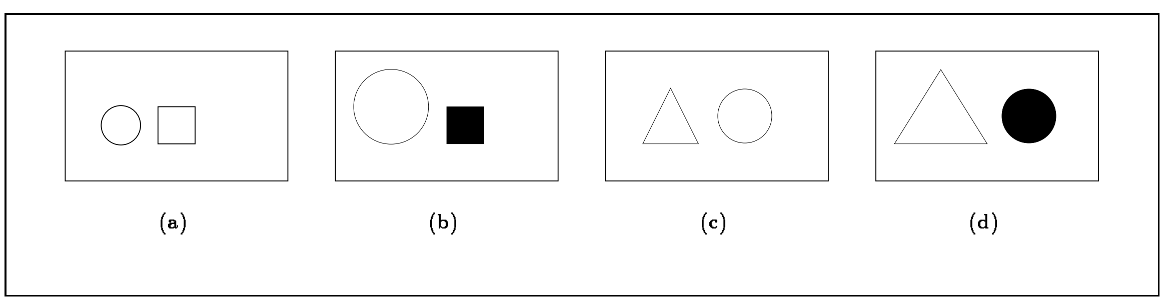


FIG. **4.20** – Exemple 2 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1],LigneDe[e2]};
B1={LigneDe[e7],LigneDe[e5]};
C1={LigneDe[e3],LigneDe[e1]};
Resolve[A1,B1,C1,App2,AppApp2] // Voir
```

```
↪ Choisir[AssocSet[FromTo[1,{1}],FromTo[2,{2}]],
  AssocSet[FromTo[1,{1}],FromTo[2,{2}]]]}

  Simultanement[
  Domaine[1,Successivement["D-elem"[PetitQ->0,GrandQ->1]],{e3}],
  Domaine[2,
  Successivement["D-elem"[CreuxQ->0,PleinQ->1]],{e1}]]}

  Choisir[{e9,e4}]
```

Justification : La transformation proposée correspond à agrandir la première figure, colorier la deuxième en noir. Ici, la figure agrandie est celle de gauche, sa triangulité n'est pas altérée.

Exemple 3

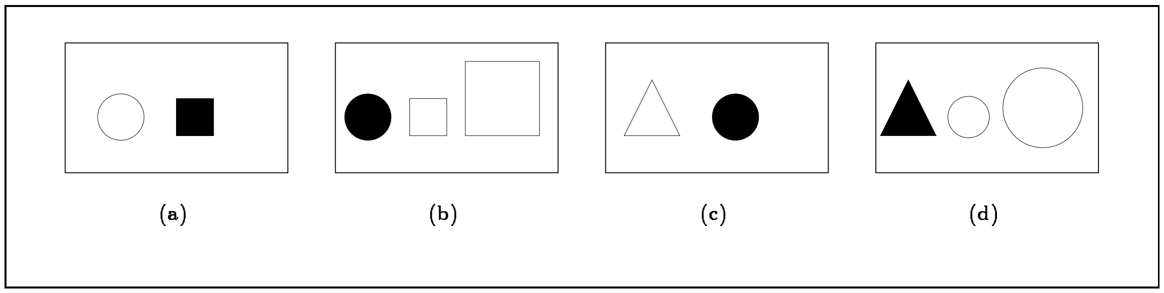


FIG. **4.21** – Exemple 3 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1],LigneDe[e5]};
B1={LigneDe[e4],LigneDe[e2],LigneDe[e8]};
C1={LigneDe[e3],LigneDe[e4]};
Resolve[A1,B1,C1,App2,AppApp2] // Voir
↪ Choisir[AssocSet[FromTo[1,{1}],FromTo[2,{2,3}]],
  AssocSet[FromTo[1,{1}],FromTo[2,{2}]]]}

  Simultanement[
  Domaine[1,Successivement["D-elem"[CreuxQ->0,PleinQ->1]],{e3}],
  Domaine[2,
  Duplique[Successivement["D-elem"[PleinQ->0,CreuxQ->1]],
  Successivement["D-elem"[PetitQ->0,GrandQ->1],
  "D-elem"[PleinQ->0,CreuxQ->1]]]],{e4}]]}

  Choisir[{e6,e1,e7}]
```

Justification : La transformation proposée correspond à noircir la première composante, éclaircir la deuxième, et la dupliquer en plus grand. Encore une fois, les propriétés de rondeur, carréïté ou triangularité ne sont pas altérées.

Exemple 4

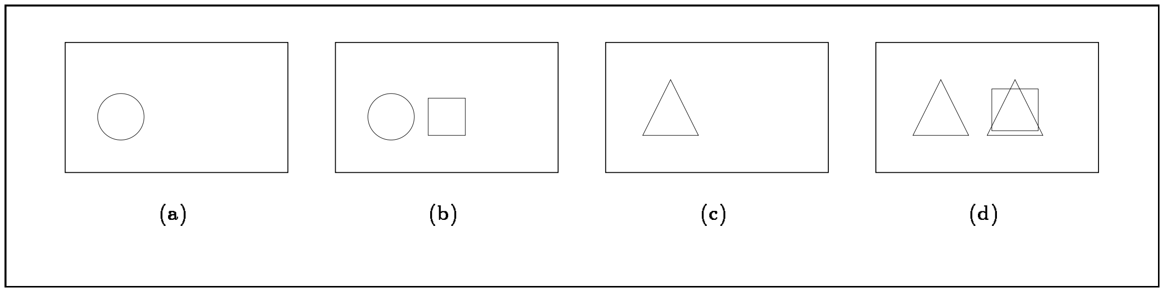


FIG. **4.22** – Exemple 4 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1]};
B1={LigneDe[e1],LigneDe[e2]};
C1={LigneDe[e3]};
Resolve[A1,B1,C1,App2,AppApp2] // Voir
↪ Choisir[AssocSet[FromTo[1,{1,2}]],AssocSet[FromTo[1,{1}]]]}

  Simultanement[
  Domaine[1,
  Duplique[Successivement["D-elem"[]],
  Successivement["D-elem"[RondQ->0,CarreQ->1]]],{e3}]]}

  Choisir[{e3,"Unknown"[False,True,True,True,False,True,False]}]
```

Justification : La transformation proposée correspond à dupliquer la figure en la rendant carrée. La triangularité n'étant pas enlevée pour autant, il y a donc production d'une solution non-stable. En effet, ici la fonction d'application de la déformation ne fait pas passer la propriété triangulaire à 0.

Exemple 5

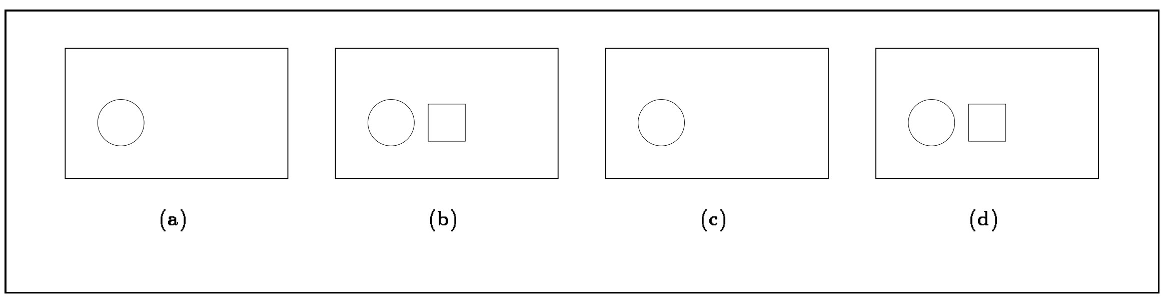


FIG. **4.23** – Exemple 5 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1]};
B1={LigneDe[e1],LigneDe[e2]};
C1={LigneDe[e1]};
Resolve[A1,B1,C1,App2,AppApp2] // Voir
```

```
↪ "appariement A-B :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement A-C :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement des appariements :
  Choisir[AssocSet[FromTo[1,{1,2}]],AssocSet[FromTo[1,{1}]]]}

  Simultanement[
  Domaine[1,
  Duplique[Successivement["D-elem"[]],
  Successivement["D-elem"[RondQ->0,CarreQ->1]]],{e1}]]}

  Choisir[{e1,e2}]
```

Justification : La transformation proposée correspond à dupliquer la figure, et rendre la deuxième carrée en lui enlevant sa caractéristique ronde. Ici, la figure C proposée est exactement égale à A, D est alors exactement égale à B.

Exemple 6

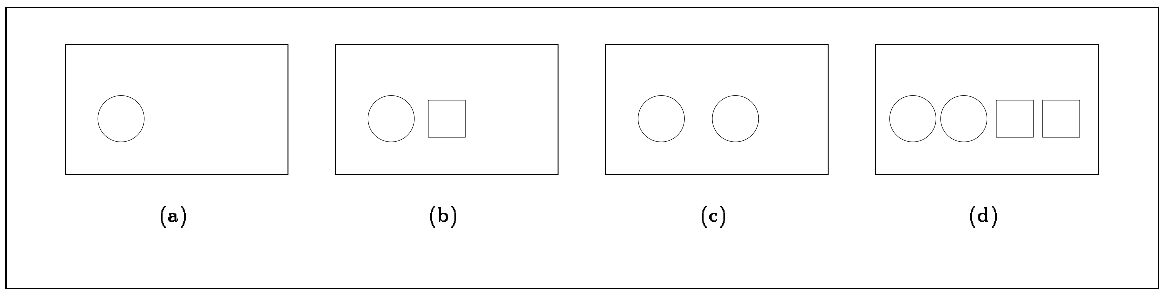


FIG. **4.24** – Exemple 6 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1]};
B1={LigneDe[e1],LigneDe[e2]};
C1={LigneDe[e1],LigneDe[e1]};
Resolve[A1,B1,C1,App1,AppApp2] // Voir
↪ "appariement A-B :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement A-C :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement des appariements :
  Choisir[AssocSet[FromTo[1,{1,2}]],AssocSet[FromTo[1,{1,2}]]]}

  Simultanement[
  Domaine[1,
  Duplique[Successivement["D-elem"[]],
  Successivement["D-elem"[RondQ->0,CarreQ->1]]],{e1,e1}]]}

  Choisir[{e1,e1,e2,e2}]
```

Justification : La transformation proposée correspond à dupliquer chaque figure en : elle-même et une deuxième carrée et moins ronde. Comme la figure de départ contient deux éléments, la solution contient quatre éléments correspondant à la duplication et transformation de chacun d'entre eux.

Exemple 7

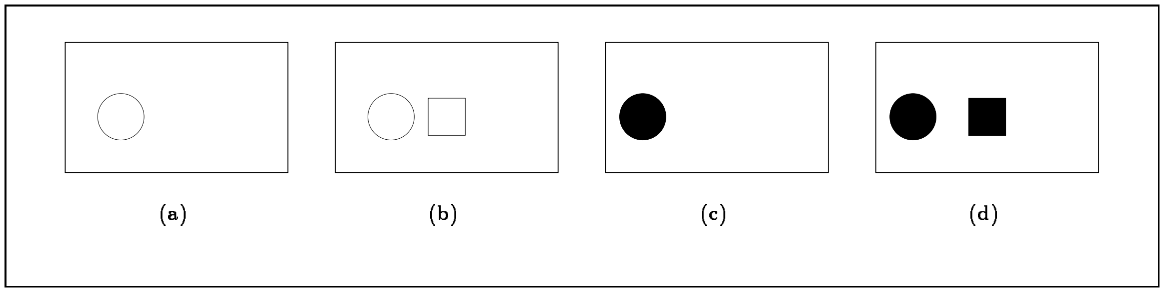


FIG. **4.25** – Exemple 7 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1]};
B1={LigneDe[e1],LigneDe[e2]};
C1={LigneDe[e4]};
Resolve[A1,B1,C1,App1,AppApp2];

↪ "appariement A-B :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement A-C :
  {AssocSet[FromTo[1,{1,2}]]}

  "appariement des appariements :
  Choisir[AssocSet[FromTo[1,{1,2}]],AssocSet[FromTo[1,{1}]]]}

  Simultanement[
  Domaine[1,
  Duplique[Successivement["D-elem"[]],
  Successivement["D-elem"[RondQ->0,CarreQ->1]]],{e4}]]}

  "solutions
  {Paire[1,{e4,e5}]}
```

Justification : La transformation proposée correspond à la même que précédemment, mais appliquée sur un élément effectivement rond, ce qui donne bien un carré, stable. La couleur noire n'est pas enlevée.

Exemple 8

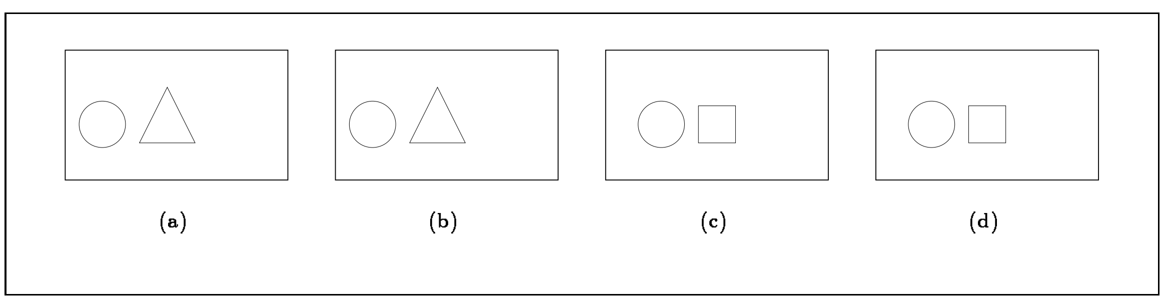


FIG. **4.26** – Exemple 8 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e1],LigneDe[e9]};
B1={LigneDe[e1],LigneDe[e9]};
C1={LigneDe[e1],LigneDe[e2]};
Resolve[A1,B1,C1,App2,AppApp2];
```

```
↪ "appariement A-B :
  {AssocSet[FromTo[1,{1}],FromTo[2,{2}]]}

  "appariement A-C :
  {AssocSet[FromTo[1,{1}],FromTo[2,{}2}]]}

  "appariement des appariements :
  Choisir[AssocSet[FromTo[1,{1}],FromTo[2,{2}]],
  AssocSet[FromTo[1,{1}],FromTo[2,{2}]]]}

  Simultanement[Domaine[1,Successivement["D-elem"[]],{e1}],
  Domaine[2,Successivement["D-elem"[]],{e2}]]}

  "solutions
  {Paire[1,{e1,e2}]
```

Justification : La transformation proposée correspond à l'identité.

Exemple 9

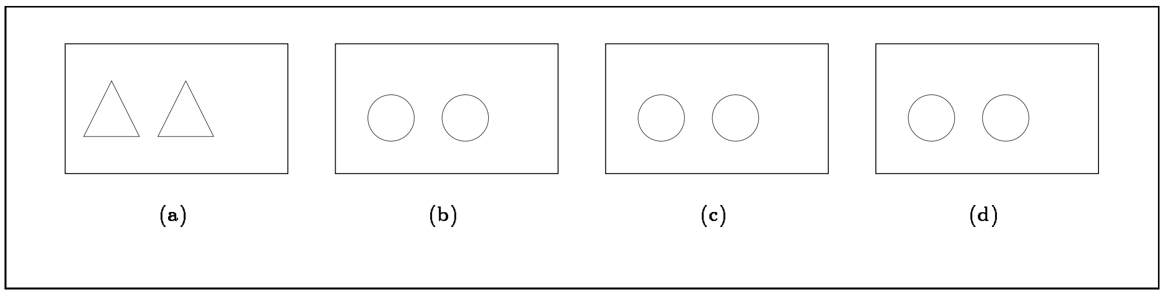


FIG. **4.27** – Exemple 9 de résolution de test de Q.I. avec ESQIMO

```
A1={LigneDe[e3],LigneDe[e3]};
B1={LigneDe[e1],LigneDe[e1]};
C1={LigneDe[e1],LigneDe[e1]};
Resolve[A1,B1,C1,App2,AppApp2];
```

```
↪ "appariement A-B :
  {AssocSet[FromTo[1,{1}],FromTo[2,{2}]]}

  "appariement A-C :
  {AssocSet[FromTo[1,{1}],FromTo[2,{2}]]}

  "appariement des appariements :
  Choisir[AssocSet[FromTo[1,{1}],FromTo[2,{2}]],
  AssocSet[FromTo[1,{1}],FromTo[2,{2}]]]}

  Simultanement[
  Domaine[1,
  Successivement["D-elem"[TriangleQ->0,RondQ->1]],{e1}],
  Domaine[2,
  Successivement["D-elem"[TriangleQ->0,RondQ->1]],{e1}]]}

  "solutions
  {Paire[1,{e1,e1}]}
```

Justification : La transformation proposée correspond à tout transformer en B.

4.5 Discussion de l'algorithme ESQIMO

Dans cette section nous allons discuter de l'approche ESQIMO pour résoudre un problème d'analogie. Remarquons tout d'abord que :

- *L'approche* ESQIMO *ne dépend pas de la nature géométrique du domaine d'application.* Nous résolvons en fait un problème abstrait d'analogie, et nous aurions pu choisir de l'illustrer dans un domaine numérique par exemple. Si nous avons choisi de travailler sur des figures géométriques, c'est surtout qu'elle permettent de saisir plus rapidement une conjonction plus grande de propriétés (forme, taille, couleur, texture, position), et sont donc plus « parlantes ». Mais nous insistons sur le fait qu'à aucun moment nous nous appuyons sur la nature géométrique du domaine d'application pour produire l'analogie.

- ESQIMO *réalise un raisonnement diagrammatique*, au sens où nous l'avons défini dans le chapitre 1. En effet, les données de la tâche à effectuer sont traduites en un espace dont l'organisation reflète la structure du problème. La résolution du problème consiste à trouver un chemin entre deux entités spatiales et à produire un chemin « similaire » entre deux autres entités. Cette similarité de chemin constitue l'analogie formelle entre les paires de configurations.

Nous avons fait apparaître dans la présentation de l'algorithme un certain nombre de choix, qui peuvent sembler arbitraire, par exemple dans la politique d'appariement. En fait, ces choix constituent autant de variantes possibles d'une même famille d'algorithmes. Nous n'avons matériellement pas eu le temps d'explorer les diverses instances de cette famille, ni de comparer leurs résultats, mais nous allons discuter les différents paramètres de cette famille :

- la description d'une configuration,
- le choix de l'unité de transformation,
- la détermination du domaine d'une transformation,
- le choix d'un chemin,
- la transformation associée à un chemin,
- la formalisation de la similitude des chemins.

Chacun de ces points constitue autant de possibilités de paramétrer l'approche ESQIMO. Nous allons les détailler ci-dessous.

4.5.1 Description d'une configuration

La description d'une configuration est faite à l'aide d'un ensemble de prédicats, suivant le cadre présenté au chapitre 2. Cet ensemble de prédicats est arbitraire, et nous pouvons donc, sans rien toucher au reste de l'algorithme, compléter les descriptions ou bien en choisir d'entièrement différentes.

Cependant, décrire une configuration uniquement en termes de prédicats sur les éléments de la configuration peut être contraignant. Prenons l'exemple, dans le domaine des figures géométriques, de la *position* d'un élément de la figure. A priori, la position n'est pas un prédicat puisqu'elle associe une coordonnée (x, y) à chaque élément de la figure. Deux solutions sont alors envisageables.

- On peut toujours trouver un codage de ce type de propriété en termes de prédicats. Par exemple, pour la position, on peut envisager autant de prédicats que d'emplacements pour un élément du graphique. Si notre figure est décrite sur un damier 4×4, il y aura 16 prédicats.

- On peut considérer un autre codage qui utilise un complexe simplicial. Par exemple, pour la position, si ce que l'on veut prendre en compte c'est en fait le voisinage d'un élément (un carré est à coté du rond), on peut coder la position comme une relation binaire (cf. chapitre 2) «e_i *VoisinsDe* e_j». Dans ce cas, le complexe codant la nouvelle propriété ne peut pas être intégré dans le complexe $C(\Omega)$. Il faut donc considérer plusieurs complexes différents pour représenter une configuration : $C(\Omega)$, $C(\Omega')$, $C(\Omega'')$, ... Cependant, sur chaque complexe, le travail à effectuer est similaire à celui réaliser sur $C(\Omega)$.

La deuxième approche généralise simplement ESQIMO en envisageant un travail parallèle sur plusieurs complexes. Le cas idéal correspond à celui où ces complexes sont sans interaction (par exemple, n'importe quelle figure peut se trouver à n'importe quelle position). Si les propriétés envisagées interagissent, alors il faut qu'elles soient codées dans le même complexe.

4.5.2 Le choix de l'unité de transformation

Dans nos exemples, nous avons considéré que l'analogie entre deux configurations dépendaient uniquement des rapports d'analogie entre les éléments de la configuration. Un élément de la configuration étant représenté par un simplexe dans un complexe, cela se traduit par la recherche d'un chemin entre deux simplexes, ce chemin représentant la transformation des éléments de départ. L'analogie se représente donc par un ensemble de chemins entre simplexes.

La situation peut se généraliser sans problème en considérant des *chemins entre complexes*. Cette généralisation s'interprète naturellement de la manière suivante : on ne représente plus cette fois la transformation d'un élément en un autre élément, mais plus généralement, la transformation d'un groupe d'éléments en un autre groupe d'éléments. On peut ainsi exprimer la transformation de la paire (carré, rond) en triangle alors que carré ne se transforme pas en triangle, pas plus que le rond. Le complexe considéré ne doit pas correspondre nécessairement à une union d'éléments. Par exemple le complexe (carré, rondeur) représente un carré en présence d'un élément rond quelconque.

Le problème à résoudre alors est dans le choix des complexes considérés. Ce problème ne fait que généraliser le problème du choix des simplexes à apparier. Restreignons nous dans la suite aux problèmes des simplexes à apparier, nos commentaires étant valables aussi si nous généralisons la situation aux complexes.

4.5.3 Détermination du domaine d'une transformation

Ce problème est celui du choix des simplexes qui vont se transformer l'un en l'autre. Ce choix se pose en fait deux fois : la première fois pour apparier des éléments de A avec des élément de B et une seconde fois pour décider quels éléments de A sont appariés avec quels éléments de C.

Nous avons traité l'appariement de manière purement combinatoire en proposant deux fonctions d'appariement possibles (les fonctions `App1` et `App2`) et deux fonctions permettant de mettre en relation les appariements $A : B$ et $A : C$ (les fonctions `AppApp1` et `AppApp2`). Les fonctions `App1` et `App2` correspondent à deux stratégies différentes pour calculer $A : B$ ou bien $A : C$. Les fonctions `AppApp1` et `AppApp2` représentent deux stratégies différentes pour combiner les appariements, comme nous allons l'expliquer.

Les fonctions `AppApp1` et `AppApp2` sont nécessaires car ESQIMO est conçu pour générer plusieurs solutions et permet donc d'associer à un simplexe s de A plusieurs appariements possibles vers B et plusieurs appariements possibles vers C. Supposons par exemple que $A : B = \{(s, s_B),\ (s, s'_B)\}$ et que $A : C = \{(s, s_C),\ (s, s'_C)\}$. Alors la transformation $s \to s_B$ peut s'appliquer à s_C ou bien à s'_C, ce qui donne lieu à deux solutions possibles. Il en va de même pour la transformation $s \to s'_B$ et donc à partir des deux ensembles d'appariements $A : B$ et $A : C$ donnés en exemple, on peut générer quatre solutions. Les fonctions `AppApp1` et `AppApp2` sont deux stratégies différentes pour choisir quelles solutions parmi ces quatre on veut générer.

Nous avons qualifié notre traitement de l'appariement de *combinatoire*. En effet, les fonctions `App1`, `App2`, `AppApp1` et `AppApp2` qui permettent d'implémenter cet appariement ne tiennent aucun compte de la nature topologique des éléments à apparier. Évidement, on peut en tenir compte, par exemple pour apparier de manière privilégiée les simplexes de même dimension, ou bien qui ont une connectivité (i.e. une ressemblance) maximale, etc. De nombreuses stratégies « diagrammatiques » sont possibles pour raffiner et guider le choix combinatoire.

4.5.4 Le choix d'un chemin

Soit les deux simplexes s_i et s_j correspondant à deux élément e_i et e_j. Les chemins d'extrémités s_i et s_j correspondent aux transformations que nous envisageons entre e_i et e_j. Il peut exister 0, 1 ou plusieurs chemins entre s_i et s_j. Le cas où il existe un seul chemin ne pose pas de problème.

Dans le cas où il n'y a pas de chemins entre s_i et s_j, nous considérons que la transformation de e_i en e_j est « impensable ». Il faudrait donc remettre en cause l'appariement qui a associé s_i à s_j (cf. l'item précédent). Nous ne traitons pas ce cas dans l'algorithme décrit.

Quand il y a plusieurs chemins entre s_i et s_j, nous avons arbitrairement choisi parmi les chemins de plus grande dimension, un des chemins les plus court. Ce n'est pas la seule stratégie possible. On peut par exemple conserver tous les chemins possibles pour étudier combinatoirement les différentes solutions qui peuvent être générées, on peut choisir un chemin parmi les plus court indépendamment de la dimension du chemin, on peut choisir un chemin en tenant compte des autres chemins à choisir, etc.

4.5.5 La transformation associée à un chemin

Nous associons une transformation à un chemin. Cette association est faite de la manière suivante : un chemin étant constitué de pas élémentaires (un pas étant deux simplexes successifs sur le chemin), la transformation est constituée de la succession des *transformations élémentaires* associées aux pas élémentaires.

Il se pose donc la question d'associer une transformation élémentaire à une paire de simplexe (s, s'). En fait, il existe de nombreuses fonctions f telles que $f(s) = s'$. Nous avons proposé quatre choix possibles d'une telle fonction, ce sont les fonctions `DefElem1` à `DefElem4`. D'autres choix sont encore possibles. On peut même mettre en cause ce passage par des transformations élémentaires pour calculer la transformation associée à un chemin. Le tout est de fournir pour chaque chemin, une transformation.

4.5.6 La formalisation de la similitude des chemins

Quand on a (enfin) obtenu un chemin, disons entre s_i et s_j, on veut considérer ce chemin comme une transformation qu'on va appliquer à s_k. On obtient en résultat un nouveau simplexe s_l. Si $s_l \in C(\Omega)$, on dit que s_l est *stable* (c'est un élément de l'univers connu) et sinon on le qualifie de « monstre » (c'est un élément nouveau). Selon l'objectif[1], on acceptera ou pas ces monstres.

Une transformation se traduit par un chemin. Il y a donc un chemin entre s_k et s_l qui est associé à la transformation. Notre idée véritable et diagrammatique est que nous pouvons nous passer de la notion de transformation pour nous intéresser directement à la similitude des deux chemins $s_i \rightsquigarrow s_j$ et $s_k \rightsquigarrow s_l$.

En effet, l'idée de transformation composée de transformations élémentaires n'est finalement qu'un mécanisme permettant d'exhiber un chemin $s_k \rightsquigarrow s_l$ présentant une « similitude de degré 0 » avec $s_i \rightsquigarrow s_j$. Cette similitude est la suivante :

- les deux chemins ont le même nombre de pas élémentaires,

1. Si l'objectif est de résoudre des tests de QI, alors le monstre correspond à une solution invalide qu'il faut rejeter, remettant en cause l'appariement dont il est issu. Par contre, si on considère le système ESQIMO comme un solveur d'équations, alors on a produit une solution qui n'était pas exprimable dans l'algèbre de départ et qui peut être utilisée pour l'étendre. Pour prendre une métaphore mathématique, à partir d'une équation qui n'impliquait que des entiers, comme par exemple $x^2 = 0$, on produit une solution qui n'est pas entière $\sqrt{2}$.

- chaque pas $(s^p_{i,j}, s^{p+1}_{i,j})$ de $s_i \rightsquigarrow s_j$ est similaire au pas $(s^p_{k,l}, s^{p+1}_{k,l})$ de $s_k \rightsquigarrow s_l$ en ce sens qu'il existe une famille de fonctions f_p tel que $f_p(s^p_{i,j}) = s^{p+1}_{i,j}$ et $f_p(s^p_{k,l}) = s^{p+1}_{k,l}$.

Nous qualifions de «degré 0» cette similitude, car elle préserve peu de propriétés topologiques entre les deux chemins. Par ailleurs, afin de rendre compte de la création possible des monstres, il faut se rendre compte que l'espace dans lequel le chemin $s_k \rightsquigarrow s_l$ n'est pas le même que l'espace dans lequel chemine $s_i \rightsquigarrow s_j$. En effet, $s_i \rightsquigarrow s_j$ prend place dans $C(\Omega)$ alors que $s_k \rightsquigarrow s_l$ est envisagé dans le complexe complet Ω (le complexe complet de Ω est le simplexe dont les sommets sont les éléments de Ω). Autrement dit, quand on envisage le chemin $s_k \rightsquigarrow s_l$ on a perdu la structure topologique de $C(\Omega)$ (cela peut avoir un intérêt quand on veut produire des monstres !).

D'autres notions de similitude sont possibles et doivent être étudiées. Nous n'avons hélas pas eu matériellement le temps d'aborder cette étude. Nous esquisserons deux approches :

- **Homotopie de chemins.** On va supposer que les deux chemins prennent place dans le même espace. Alors une notion de similitude naturelle serait que les chemins soit *homotopes* (au sens combinatoire, cf. chapitre 2). Autrement dit, on peut déformer dans $C(\Omega)$ le chemin $s_i \rightsquigarrow s_j$ en chemin $s_k \rightsquigarrow s_l$. Pour déterminer s_l, il faut considérer que ce doit être une extrémité commune aux chemins $s_k \rightsquigarrow s_l$ homotopes à $s_i \rightsquigarrow s_j$ mais aussi aux chemins $s_j \rightsquigarrow s_l$ homotopes aux chemins $s_i \rightsquigarrow s_k$ (on retrouve ici l'idée de symétrie du problème $A : B :: C : D$ qui amène à regarder *aussi* les rapports $A : C$).

- **Cobordisme.** On peut considérer les deux chemins comme une chaîne. On peut alors les voir comme les bords d'un complexe inclus dans $C(\Omega)$ (cf. chapitre 2). On doit donc trouver un complexe dont $(s_i \rightsquigarrow s_j) + s_k$, vu comme une chaîne, fait partie du bord. Le reste du bord constitue une chaîne associée à un chemin $s_k \rightsquigarrow s_l$ (privé de s_k). Il reste à trouver le bon s_l dans cette chaîne (on peut donc aussi faire intervenir $A : C$ comme évoqué ci-dessus).

Ces approches ne sont qu'esquissées mais nous aimerions dans la suite du travail pousser leur étude.

Chapitre 5

Conclusions et perspectives

5.1 Résumé des travaux

Dans ce rapport nous avons défendu la conception et le développement d'outils spécifiques pour la modélisation des représentations spatiales. La distinction entre les relations spatiales de nature purement topologique et celles de nature métrique a motivé la notion de *représentation et de raisonnement diagrammatique* (RRD). Nous avons défini la représentation diagrammatique comme la capacité à élaborer et mémoriser une représentation impliquant des relation spatiales topologiques, et le raisonnement diagrammatique comme la capacité à structurer et parcourir mentalement cet espace topologique.

Notre motivation est que les relations spatiales permettent d'organiser et de structurer la représentation de certaines connaissances, de manière à ce qu'une opération spatiale sur la représentation corresponde alors à une opération ayant une sémantique naturelle dans le contexte des connaissances à représenter.

Dans ce cadre, notre objectif est de développer des outils effectifs pour modéliser et automatiser les différentes formes de RRD. Notre thèse est que certaines notions de topologie algébrique permettent de clarifier et de formaliser les concepts intervenant en RRD.

En effet, les modèles utilisés traditionnellement en Intelligence Artificielle pour la représentation des connaissances (logique, réseaux sémantiques) sont mal adaptés à l'expression et au traitement des relations spatiales topologiques. La recherche d'outils formels plus adéquats s'impose donc. La topologie est un candidat naturel, et la topologie algébrique combinatoire sur les ensembles finis est à même d'offrir des outils effectifs (c'est-à-dire des programmes qui tournent sur un ordinateur).

Concrètement, nous avons proposé d'utiliser la notion de complexe simplicial abstrait (CS). Nous avons mis en œuvre cette notion dans deux applications : une tâche de catégorisation et un problème d'analogie.

Notre objectif avec la première application est limité à montrer qu'il est possible d'extraire à peu de frais une représentation diagrammatique (un CS) à partir d'une procédure classique de catégorisation. Le résultat du processus de catégorisation fournit un CS qu'il est possible de représenter sous forme de treillis : on obtient alors une ontologie des catégories extraites. La tâche de catégorisation que nous avons décidé de représenter et de traiter diagrammatiquement, souscrit à un modèle général développé par Holland et *al.* Ce modèle, qui ne fait pas spécialement référence à une approche spatiale du raisonnement, fournit aussi un cadre pour définir l'analogie.

L'application suivante est un exemple de raisonnement analogique qu'on peut trouver à l'œuvre dans certains tests de QI : il s'agit de compléter une suite trois figures A, B, C, par une quatrième figure D, de sorte que D soit à C ce que B est à A. La résolution par des méthodes topologiques de ce problème, noté $A : B :: C : D$, a donné lieu au développement du logiciel **ESQIMO**. Le principe d'ESQIMO est de représenter la transformation $A : B$ de A en B par un *chemin* dans un certain espace. La construction du chemin $A : D$, qui doit être similaire au chemin $A : B$, permet de produire la figure D recherchée. Nous avons utilisé une notion élémentaire de similitude de chemin, faute d'avoir eu le temps d'étudier des notions de similitude impliquant des outils plus élaborés de topologie algébrique, comme l'homotopie ou l'homologie.

5.2 Perspectives

Les deux applications que nous avons abordé valident la faisabilité de notre approche : il est possible de mettre en œuvre des notions de topologie algébrique pour modéliser et automatiser certaines opérations cognitives. Beaucoup de travail est encore nécessaire pour poursuivre les deux études commencées et nous avons évoqué à la fin du chapitre 4 toutes les perspectives ouvertes par le logiciel ESQIMO. Mais nous espérons que les résultats partiels que nous avons présentés ont convaincu de lecteur du grand intérêt de l'approche topologique. Certes, le caractère nouveau et formel peut peut-être freiner dans un premier temps l'utilisation de tels outils, mais nous sommes persuadés cependant que le caractère très intuitif des concepts est un argument puissant en faveur de leur utilisation.

Nous désirons poursuivre ce travail en étudiant plus systématiquement :

- la représentation de connaissances déclaratives *et* procédurales à l'aide de complexes simpliciaux ;
- l'apport spécifique de ce type de représentation topologique vis à vis des modèles informatiques de représentation des connaissances (modèles logiques, modèles objets, réseaux sémantiques et graphes conceptuels, etc.) ;
- la formalisation par la topologie d'une notion d'analogie ;
- la conception et le développement d'outils d'assistance au raisonnement diagrammatique, comme par exemple la synthèse d'architecture de systèmes (matériel ou logiciel).

Ce dernier point découle naturellement des analyses esquissées dans les chapitres 3 et 4. En effet, dans le chapitre 3 nous avons montré comment extraire automatiquement des «patterns» et dans le chapitre 4 nous avons montré comment on peut construire des patterns par analogie avec d'autres patterns. Notre idée est d'appliquer cette « technologie » à un domaine plus structuré comme la conception d'architectures logicielles ou matérielles.

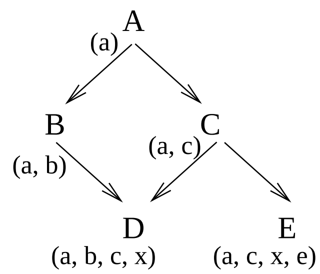
La conception ou l'analyse d'un système passe souvent par des diagrammes permettant d'expliciter l'architecture du système : graphes à flot de données, actigrammes et datagrammes SADT, cartes fonctionnelles, etc. Ces diagrammes sont manipulés à travers un ensemble de diagrammes de base et de structures de composition réduites. Par exemple, dans le cas des circuits électriques, les éléments du circuit se composent uniquement en série ou en parallèle. Pour analyser ou concevoir un nouveau circuit, l'électronicien dispose de schémas types (amplificateur, inverseur, filtre, etc.) qui lui permettent l'analyse ou la synthèse de nouveaux circuits par analogie avec ces schémas. En fait, la conception et le développement de systèmes repose en grande partie sur l'analogie avec des "patrons" existants (***design patterns*** en anglais) que ce soit pour des systèmes matériels (architecture des bâtiments [AIS$^+$77], électronique...) ou logiciels [GHJV96].

Dans le cas de la conception orientée objet de programme, qui nous intéresse plus particulièrement, les relations qui sont intéressantes à représenter sont les relations entre classes, méthodes et slots.

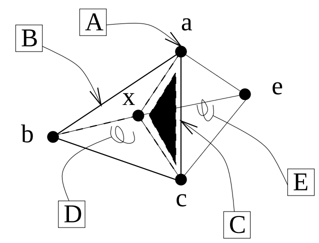
- La relation d'héritage est une relation entre classes. Une autre relation entre classes qui est intéressante est celle du sous-typage (l'héritage qui est une méthode de construction de classe, implique le sous-typage, mais l'inverse n'est pas vrai) ;
- Les relations d'appartenance des slots ou des méthodes aux classes, sont utiles pour analyser le sous-typage ;

– L'accès à un slot, ou l'appel à une méthode, par une méthode donnée, est un exemple de relation entre les slots et les méthodes et entre les méthodes et les méthodes.

Toutes ces relations peuvent s'analyser en termes de complexes simpliciaux. La figure 5.1 montre par exemple le complexe correspondant à la relation d'appartenance des slots aux classes, pour une hiérarchie donnée. L'analyse de ces relations est indispensable si l'on veut, par exemple, structurer automatiquement un logiciel, afin de maximiser la réutilisation, ou de rendre ce logiciel plus évolutif. Par exemple, nous pourrions générer automatiquement des classes intermédiaires correspondant aux propriétés minimales nécessaires pour la définition d'une méthode, ce qui permet d'exporter un maximum de fonctionnalités avec une structure de classe minimale.



(a) Hiérarchie de classes avec les slots



(b) Représentation sous forme de complexe simplicial correspondante

FIG. **5.1** – Hiérarchie dans les LOO, et sa représentation sous forme de complexe: les slots sont en minuscules et les classes en majuscules. Sur le complexe simplicial associé, les sommets sont les slots et les simplexes sont désignés par le nom de la classe qu'ils représentent dans une étiquette. La face commune aux simplexes représentant D et E, est noircie, elle correspond au simplexe $\langle a, c, x \rangle$

Une autre application possible de la technologie esquissée dans les chapitres 2 et 3, est d'extraire automatiquement des « patterns de codage », c'est-à-dire des éléments réutilisables de logiciels. Le langage des complexes nous semble en effet particulièrement apte à formaliser la description des «design pattern» [GHJV96, Nie93, BMR+96]. La figure 5.2 donne un exemple de pattern extrait de [BMR+96], page 79. Un langage de patterns étant formalisé grâce aux CS, nous pouvons espérer les extraire automatiquement, et raisonner dessus, par exemple par analogie, à la manière du système ESQIMO, afin de synthétiser de nouvelles architectures.

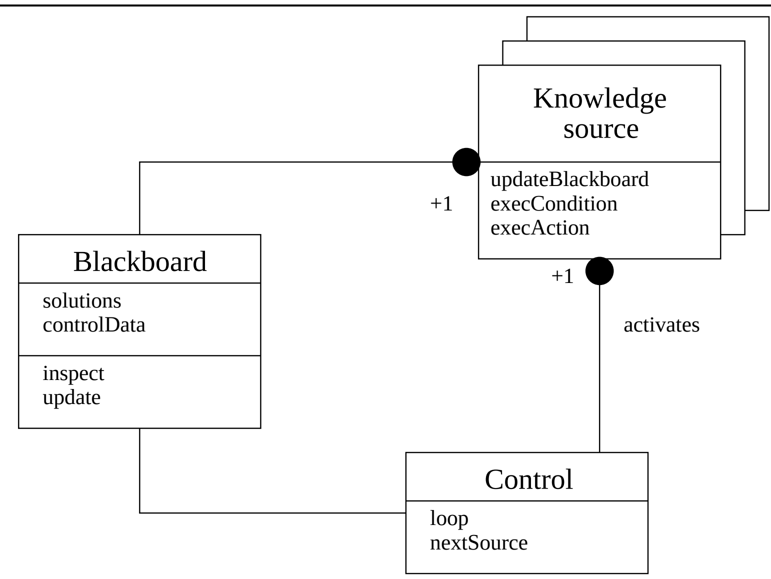


FIG. **5.2** – Exemple de pattern [BMR+96]

Annexe A

Théorie de l'homologie

Les propriétés algébriques des groupes d'homologie doivent refléter les propriétés topologiques de l'espace topologique d'origine. Afin de préciser dans quelle mesure, nous allons commencer par introduire quelques notions nécessaires au calcul des bords d'un complexe et des nombres de Betti.

A.1 Groupes d'homologie d'un complexe

Dans la suite, Γ est un complexe simplicial tel que défini dans le corps du rapport.

Définition 9 (Idemgroupe) *Un* idemgroupe *est un groupe pour lequel, tous les éléments vérifient :*

$$x + x = 0$$

c'est-à-dire qu'ils vérifient : $x = -x$.

A.1.1 Bords et calcul d'incidences

Nous allons définir une notion particulière de bord qui vérifie les propriétés formelles données au chapitre 2.

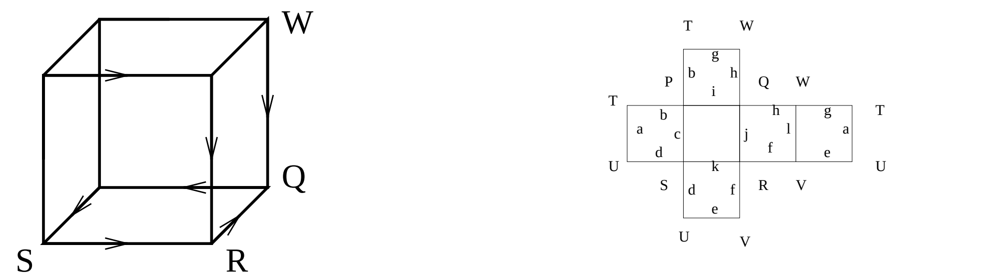
Définition 10 (Bord) *Soit C une k-chaîne de Γ. Le* bord *de C, noté $\partial(C)$, est la $(k-1)$-chaîne construite à partir des $(k-1)$-simplexes incidents un nombre impair de fois dans le complexe Γ.*

Afin de calculer le bord d'un complexe, nous devons déterminer les incidences des simplexes d'un complexe. Cela s'effectue à l'aide des coefficients d'incidence des simplexes de Γ :

Définition 11 (Coefficient d'incidence) *Soit x un k-simplexe et y un $(k+1)$-simplexe de Γ. Le* coefficient d'incidence *de x dans y est le nombre de fois où x apparaît dans le bord de y.*

Prenons des exemples. Considérons le cube de la figure A.1(a), équivalent à une sphère. Nous l'avons déroulé et nous avons nommé tous les sommets et toutes les arêtes sur le schéma de la figure A.1(b).

Chacun des sommets est un 1-simplexe et nous allons compter combien de fois ils apparaissent dans les 2-simplexes que sont les arêtes. Ce calcul est reporté sur la table suivante :



(a) Complexe associé à une sphère. Les sommets ne sont pas tous nommés pour ne pas encombrer la figure.

(b) Complexe associé à une sphère déplié, les arêtes sont nommées.

FIG. **A.1** – Complexe associé à une sphère et complexe déplié pour le calcul des incidences

	P	Q	R	S	T	U	V	W
a	0	0	0	0	1	1	0	0
b	1	0	0	0	1	0	0	0
c	1	0	0	1	0	0	0	0
d	0	0	0	1	0	1	0	0
e	0	0	0	0	0	1	1	0
f	0	0	1	0	0	0	1	0
g	0	0	0	0	1	0	0	1
h	0	1	0	0	0	0	0	1
i	1	1	0	0	0	0	0	0
j	0	1	1	0	0	0	0	0
k	0	0	1	1	0	0	0	0
l	0	0	0	0	0	0	1	1

Nous voyons, par exemple, que le sommet T participe aux 2-simplexes a, b, et g. Nous ferions de même en numérotant les faces du cube et en comptant combien de fois chaque arête apparaît dans chaque face.

Prenons un deuxième exemple, dans le tore de la figure A.2, le sommet P, apparaît deux fois comme bord des arêtes a et b.

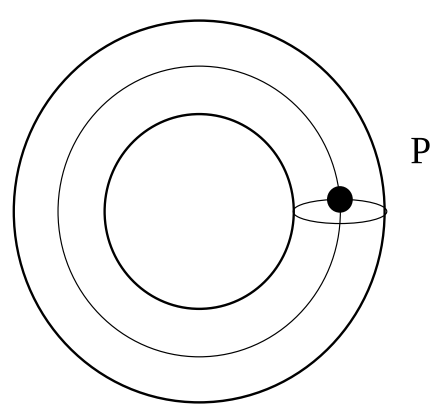


FIG. **A.2** – Tore

L'opérateur de bord est additif :

$$\partial(c_1 + c_2) = \partial(C_1) + \partial(C_2)$$

Définition 12 (k-cycle) *Un* k-cycle $(k = 0, 1)$ *est une k-chaîne avec un bord nul. Par convention, toutes les 0-chaînes sont appelées des 0-cycles. Les k-cycles, $Z_k(\Gamma)$ forment un groupe.*

Définition 13 (k-bord) *Un* k-bord $(k = 0, 1)$ *est une k-chaîne qui est le bord d'une $(k + 1)$-chaîne. Les k-bords, $B_k(\Gamma)$ forment un groupe.*

Théorème 2 *Tout bord est un cycle.*

A.1.2 Calcul des groupes d'homologie

Définition 14 (Chaînes homologues) *Deux k-chaînes $(k = 0, 1, 2)$, C_1 et C_2, sont* homologues, *noté $C_1 \sim C_2$, ssi $C_1 + C_2$ est un k-bord.*

La relation d'homologie possède les propriétés suivantes :

$$\begin{array}{ll} (a) & C \sim C, \\ (b) & C_1 \sim C_2 \Rightarrow C_2 \sim C_1. \\ (c) & C_1 \sim C_2 \quad \text{et} \quad C_2 \sim C_3 \Rightarrow C_1 \sim C_3. \\ (d) & C_1 \sim C_2 \quad \text{et} \quad C_3 \sim C_4 \Rightarrow (C_1 + C_3) \sim (C_2 + C_4). \end{array}$$

L'homologie est donc une relation d'équivalence et l'on peut considérer le groupe $Z_1(\Gamma)/\sim$. Le nouveau groupe, noté $H_k(\Gamma)$ et appelé le k-ième groupe d'homologie de Γ.

Définition 15 (Triangulation) *Une triangulation d'un polyèdre P est une subdivision de P en un nombre fini de triangles, de sorte que chaque bord de P soit le bord d'un seul triangle de la subdivision, et chaque arête de l'intérieur de P, soit le bord d'exactement deux triangles de la subdivision.*

Théorème 3 (Invariance) *Soit Γ un complexe, composé de simplexes de dimension 2 ou moins. Les groupes d'homologie de Γ sont indépendants du choix de la triangulation.*

A.2 Calcul des nombres de Betti

Définition 16 (Dépendance) *Soit G un* idemgroupe. *Soit $A = \{y_1, y_2, ..., y_n\}$ un sous-ensemble de G. Un élément x de G* dépend *de A si x est une combinaison linéaire d'éléments de A :*

$$x = a_1 y_1 + a_2 y_2 + ... + a_n y_n$$

où les $a_i \in \{0, 1\}$.

Définition 17 (Base) *Soit G un idemgroupe et A un sous-ensemble de G. Le sous-ensemble A engendre G si chaque élément de G dépend de A. De plus, A est indépendant (ou libre) si aucun élément de A ne dépend d'un autre élément de A. Un sous-ensemble indépendant et générateur est une* base *de G.*

Théorème 4 *Tout sous-ensemble A générateur de G contient un sous-ensemble B, qui est une base de G.*

Théorème 5 *Deux bases quelconques de G ont le même nombre d'éléments.*

Définition 18 (Rang) *Le nombre d'éléments dans une base de G s'apelle le* rang *de G.*

Considérons le groupe des k-chaînes $C_k(\Gamma)$. Chaque chaîne étant une somme de simplexes et l'ensemble des simplexes étant libre, le groupe des chaînes constitue une base de Γ. Le rang de cette base est appelé c_k.

Passons maintenant au groupe d'homologie de Γ, $H_k(\Gamma)$. Le rang de $H_k(\Gamma)$ est appelé h_k. En vertu du théorème d'invariance, ce nombre doit être une mesure d'une propriété topologique du complexe. Ces nombre h_k sont appelés les *nombres de Betti* de Γ.

Le 0^e nombre de Betti, h_0 est toujours égal à 1 pour tout complexe connecté, puisque deux arêtes sont alors toujours homologues. Plus généralement, une surface quelconque est toujours une union de parties connectées, appelées *composantes.* Le 0-ième nombre de Betti d'une surface quelconque est donc le nombre de ses composantes.

Le premier nombre de Betti, h_1, est le *nombre de connectivité* d'une surface. Cela nous donne le nombre maximal de courbes fermées que l'on peut dessiner sur la surface sans la diviser en deux ou plus. Par exemple, $h_k = 0$ pour la sphère.

Annexe B

Outils implémentés en Mathematica pour les complexes simpliciaux

B.1 Création des complexes

```
CreerSimplex[cSimplex[l_List, _]] := CreerSimplex[l]
CreerSimplex[l:{_cSimplex..}, c_] := cSimplex[Union@@(#[[1]]&/@l), c]
CreerSimplex[l_List] := CreerSimplex[l, void]

CreerSimplex[l_List, c_] := cSimplex[Union[l], c]
CreerSimplex[cSimplex[l_List, _], c_] := cSimplex[l, c]
CreerSimplex[l__] := CreerSimplex[{l}]

SortSimplexFct[cSimplex[l1_List, _], cSimplex[l2_List, _]] :=
                        Length[l1] > Length[l2]

CreerComplex[c:cComplex[l___], s_cSimplex] :=
          If[DansCQ[s, c], c, Union[cComplex[s, l]]]

CreerComplex[l:{_cSimplex...}] :=
          With[{lc = Union[Color/@l]},
          With[{ll = Sort[Cases[l, cSimplex[_, #]], SortSimplexFct]& /@ lc},
                   Fold [Fold[CreerComplex, #1, #2]&, cComplex[], ll]
                   ]]

CreerComplex[l:_cSimplex..]  := CreerComplex[{l}]
CreerComplex[l:_cComplex..]  := CreerComplex@@Join[l]

RandomComplex[dim_Integer, nsimp_Integer, nv_Integer] :=
          CreerComplex@@Table[RandomSimplex[Random[Integer, {0,dim}], nv], {nsimp}]

RandomComplexV[dim_Integer, nsimp_Integer, nv_Integer] /;nv≤26:=
          RandomComplex[dim,nsimp,nv] /.
                        {x_Integer:> FromCharacterCode[ToCharacterCode["A"]+x]}
```

B.2 Représentation des complexes

```
ShowComplex[c_cComplex /; Dim[c] ≤3] :=
     With[{som = Sommets[c]},
     With[{renumerote = Table[som[[i+1]] -> i, {i, 0, Length[som] - 1}]},
        {Graphics3D[Sort[List@@c, SortSimplexFct] /.  renumerote] /.  cSimplex ->
GS,
          Graphics3D[Map[Text[StringForm["‘‘", #],
          With[{ls= #/.renumerote}, PtCoord[ls] + Ecart[ls]]]&,
              som]]}
     ]]
ShowComplex[c_cComplex /; Dim[c] >3]:=
  Error["ShowComplex:  not a tridimensional complex:  ", c]



GS[] := {PointSize[0.03]}
GS[{p1_Integer}, c_] :=
   {GraphicColor[c], PointSize[0.03], Point[PtCoord[p1]]}
GS[{p1_Integer, p2_Integer}, c_] :=
    {GraphicColor[c], Thickness[0.01], Line[{PtCoord[p1], PtCoord[p2]}]}
GS[{p1_Integer, p2_Integer, p3_Integer}, c_] :=
    {GraphicColor[c], Polygon[{PtCoord[p1], PtCoord[p2], PtCoord[p3]}]}
GS[{p1_Integer, p2_Integer, p3_Integer, p4_Integer}, c_] :=
    {GraphicColor[c], Polygon[{PtCoord[p1], PtCoord[p2], PtCoord[p3],
           PtCoord[p1], PtCoord[p2], PtCoord[p4],
           PtCoord[p1], PtCoord[p4], PtCoord[p3],
           PtCoord[p4], PtCoord[p2], PtCoord[p3]}]}

PtCoord[p_] := Switch[Mod[p, 3],
                    0, {0, 0, 0},
                    1, {1, 0, 0},
                    2, {0, 0, 1}] + {0, Quotient[p, 3], 0}

Ecart[p_] := Switch[Mod[p, 3],
                    0, {-0.1, 0, -0.1},
                    1, {0.1, 0, 0},
                    2, {-0.1, 0, 0.1}]

BasicColorRule = {void -> GrayLevel[0],
                     vert -> RGBColor[0, 1, 0],
                     rouge -> RGBColor[1, 0, 0],
                     bleu -> RGBColor[0, 0, 1],
                     gris ->  GrayLevel[0.8]};

ColorRule = BasicColorRule;

ListOfColor := ColorRule /.  {Rule[x_ , y_] -> x};
GraphicColor[x_ /; MemberQ[ListOfColor, x]] := x /.  ColorRule
GraphicColor[_] := {}
```

```
ViewComplex[c_cComplex, som_List] :=
     ViewComplex[List@@c, Join[som, Complement[Sommets[c], som]]]

ViewComplex[c_cComplex] := ViewComplex[List@@c, Sommets[c]]

ViewComplex[l_List, som_List] :=
    With[{ll = Sort[l, SortSimplexFct],d = Max@@(Dim/@l)},
    With[{len = Length[som]},
    With[{renumerote = Table[som[[i+1]] -> i, {i, 0, len-1}]},
          {Graphics[ll /.  renumerote /.  cSimplex -> VS[d, len]],
         Graphics[
          Map[Text[
              StringForm["``",#],(1+0.65 d) CiCoord[len, #/.renumerote]]&,
                      som]]}

ViewSimplexList[l_List] := ViewComplex[l,Sommets[l]]

VS[d_Integer, n_Integer][{p1_Integer}, c_] :=
    {GraphicColor[c],PointSize[0.03], Thickness[0.005],
        Point[CiCoord[n, p1]],Line[{CiCoord[n, p1], (1 + 0.55 d) CiCoord[n, p1]}]}

VS[d_Integer, n_Integer][{p1_Integer, p2_Integer}, c_] :=
    {GraphicColor[c],AbsoluteThickness[2],Line[{CiCoord[n, p1],CiCoord[n, p2]}]}

VS[d_Integer, n_Integer][l_List, c_] :=
    {RandHue[c, (Length[l] - 2)/(d - 1)],
                                  Polygon[ (1 + 0.45 Length[l]) Map[CiCoord[n, #]&,
l] ]}

CiCoord[n_Integer, i_Integer] := {N[Cos[i 2 Pi / n]], N[Sin[i 2 Pi / n]]}
RandHue[c_, x_] := If[c === void, Hue[Random[Real, x]], GraphicColor[c]]
Nom[c_cComplex][l_List]:=
    With[{nm=Cases[c,cSimplex[l,_]:>l]},
        If[nm==={},
                If[Length[l]===1,First[l],""],Plus@@nm]]

DisplayComplex[c_]:=
    With[{ls=Sommets/@(List@@c)},
        With[{somTreillis=Union[Flatten[PartieDe/@ls,1]]},
            With[{label=Nom[c]/@somTreillis},
                ShowLabeledGraph[
                    HasseDiagram[MakeGraph[somTreillis,
                        ((Intersection[#2,#1]===#1)&&(#1 != #2))&]],
                    label ]
        ]]]
```

B.3 Observateurs sur les complexes

```
Color[cSimplex[_, c_]] := c

Sommets[cSimplex[l_, _]] := l
Sommets[c_cComplex] := Union@@(Sommets /@ c)
Sommets[l_List] := Union@@(Sommets /@ l)

SommetsC[cSimplex[l_,col_]] := Union[{#,col}&/@l]
SommetsC[c_cComplex] := Union@@(SommetsC /@ c)
```

```
DansSQ[cSimplex[l1_, _], cSimplex[l2_, _]] :=
                (Length[l1]≤Length[l2])&&(Complement[l1, l2] === {})
DansCQ[s1_cSimplex, s2_cSimplex] :=
   (Color[s1] === Color[s2]) && DansSQ[s1, s2]

DansSQ[s_cSimplex, c_cComplex] := LazyOr[DansSQ[s, #]&, c]
DansCQ[s_cSimplex, c_cComplex] := LazyOr[DansCQ[s, #]&, c]

DansSQ[c1_cComplex, c2_cComplex] := LazyAnd[DansSQ[#, c2]&, c1]
DansCQ[c1_cComplex, c2_cComplex] := LazyAnd[DansCQ[#, c2]&, c1]

Dim[cSimplex[l_, _]] := Length[l] - 1
Dim[cComplex[]] := -1
Dim[c_cComplex] := Max@@(Dim/@c)

Squelette[i_Integer][cSimplex[l_List, c_]] := Map[CreerSimplex[#, c]&,
SousEnsembles[i+1][l]];
Squelette[i_Integer][x_cComplex] := CreerComplex[ Flatten[Squelette[i]/@(List@@x)]];
```

B.4 Comparaison par matching

```
FindMatchS[c_cComplex, p_cComplex, var_List]:=FindMatch[c,p,var,DansSQ]
FindMatchC[c_cComplex, p_cComplex, var_List]:=FindMatch[c,p,var,DansCQ]

FindMatch[c_cComplex, p_cComplex, var_List,test_] :=
    With[{sc = Sommets[c],sp=Sommets[p]},
    With[{fixe=Complement[sp,var]},
    With[{candidat=Complement[sc,fixe]},
    With[{subs=Appariement[var,candidat]},
                    Union[ Select[Substitution[#,p]&/@subs,test[#,c]&] ]
    ]]]]

Appariement[var_List, candidat_List]:=
    If[Length[var]>Length[candidat],{},Appariement[var,candidat,{{}}]]
Appariement[{},candidat_List,rep_List]:=rep
Appariement[{x_,l___},candidat_List,rep_List]:=
    With[{subs=Appariement2[x,candidat]},
        Join@@Map[Appariement[{l},#[[2]],Map[AddSubs[#[[1]]],rep]]&,subs]]
AddSubs[x_][l_List]:=Join[{x},l]
Appariement2[x_,l_List]:=MapIndexed[{x->#1,Delete[l,#2[[1]]]}&,l]
Substitution[s_List,c_cComplex]:=
    (c/.s)/.{cComplex->CreerComplex,cSimplex->CreerSimplex}
```

B.5 Implémentation pour la q-connectivité

```
Sharing[s1_cSimplex, s2_cSimplex]:=
Intersection[CreerSimplex[s1], CreerSimplex[s2]]
Sharing[c1_cComplex,c2_cComplex]:= Intersection[Flatten[c1],Flatten[c2]]
Lconnect[s1_cSimplex,s2_cSimplex]:= Dim[Sharing[s1,s2]]
Lconnect[c1_cComplex,c2_cComplex]:= Dim[Sharing[c1,c2]]
```

```
Connections[d_Integer, s_cSimplex,c_cComplex]:=
    Select[List@@c,(d≤Dim[Intersection[s,#]])&]

CheminsQ[l_Integer,d_Integer, s1_cSimplex,s2_cSimplex,c_cComplex]:=
    CheminsQ[l-1,d,Select[List@@c,DansCQ[s1,#]&],s2,c]

CheminsQ[0,_,_,_,_]:=False
CheminsQ[_,_,{},_,_]:=False

CheminsQ[l_Integer,d_Integer,r_List,sf_cSimplex,c_cComplex]:=
    With[{voisins=Flatten[Connections[d,#,c]&/@r,1]},
        With[{target=Select[voisins,DansCQ[sf,#]&]},
            Print["voisins=",voisins];
            Print["target=",target];
            If[(target==={}),CheminsQ[l-1,d,Complement[voisins,r],sf,c],
                True]]]

CheckChemin[n_Integer,s1_cSimplex][s2_cSimplex]:=If[DansCQ[s1,s2],{n,s2},{}]

Chemins[l_Integer,d_Integer, s1_cSimplex,s2_cSimplex,c_cComplex]:=
    With[{voisins=Flatten[CheckChemin[0,s1]/@(List@@c),1]},
    With[{target=Select[voisins,DansCQ[s2,#[[2]]]&]},
            If[target==={},Chemins[0,l,d,s2,c,voisins,voisins],target]]]

Chemins[_,0,_Integer,_cSimplex,_cComplex,_List,_List]:={}
```

B.6 Extraction d'une base de percepts

```
Clear[Fragments, Diff, SimplexBase, Coord, O2C, DecLineaire];

Diff[l1_,l2_] :=
    With[{inter=Intersection[l1,l2]},
        {inter,Complement[l1,inter],Complement[l2,inter]}]

Fragments[l_List] :=
    With[{l2 = Flatten[Outer[Diff, l, l, 1], 2]},
    With[{l3 = Select[l2,(# =!= {})&]},
        Union[l3]]]

SimplexBase[l_List] := (CreerSimplex@@#)& /@ SimplexBase[Fragments[l],{}]

SimplexBase[{},r_List] := r
SimplexBase[{x_,l___}, r_List] :=
    With[{reste = Complement[x,Sequence@@r]},
        SimplexBase[{l}, If[reste==={},r,Join[r,{reste}]]]]

Coord[base_List][x_List] :=
    CreerComplex@@Select[base, (Complement[Sommets[#], x]==={})&]

DecLineaire[base_List][x_List] :=
  Select[base, (Complement[Sommets[#], x]==={})&]
```

```
ClearAll[Ajoute, Incremente]

Ajoute[base_List, image_List] :=
    With[{decomp = Sommets/@(DecLineaire[base][image])},
     With[{projo=  decomp },
        With[{ reste= Complement[image,Union[Join@@projo]]},
            If[reste==={},base,Join[base,{CreerSimplex[reste]}]]]]]

Incremente[hist_List] := Fold[Ajoute,{},hist]
```

B.7 Codage du Petit Chaperon Rouge

```
image1 ={"Chaperon","Adulte", "parler", "maison"};
image2 ={"Chaperon","Adulte", "donner","panier", "maison"};
image3 ={"Chaperon", "marcher", "arbres","panier"};
image4 = {"Chaperon", "Loup", "parler", "arbres","panier"};
image5 ={"Chaperon", "marcher", "arbres","panier"};
image6 ={"Loup", "marcher", "arbres"};
image7 = {"Adulte","dormir","maison","lit"};
image8 = {"Adulte", "Loup", "parler","maison","lit"};
image9 ={"Adulte", "Loup", "manger","maison","lit"};
image10 ={"Chaperon", "parler","maison","panier","lit"};
image11 ={"Chaperon", "Loup", "manger","maison","panier","lit"};

cr = {image1, image2, image3, image4,image5, image6, image7, image8, image9,
image10, image11};

codage1= {
"Adulte" -> {"vivant", "agréable"},
"arbres" ->{"vivant", "lieu", "méchant", "plusieurs"},
"Chaperon"->{"vivant", "agréable","petit"},
"donner"->{"moteur", ".->"},
"dormir" ->{"moteur","agréable"},
"lit" -> {"lieu", "agréable"},
"Loup" -> {"vivant","animal","petit"},
"maison"-> {"lieu"},
"manger"->{"moteur", ".<-", "agréable"},
"marcher"->{"moteur"},
"panier"->{"petit"},
"parler"->{"moteur", ".->"}
};

cr1 = Flatten /@(cr /.  codage);
```

Annexe C

Implémentation du système ESQIMO

```
ClearAll[LazyOr];
SetAttributes[LazyOr,HoldRest]
LazyOr[f_,l_]:=ReleaseHold[Or@@f/@Hold@@l]

ClearAll[e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13];
ClearAll[Univers];
Univers = { e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12};
ClearAll[PleinQ,CreuxQ,RondQ,CarreQ,triangleQ, GrandQ, PetitQ, ListeProp,
    pX1,pX2,pX3,pY1,pY2,pY3];
ListeProp = {RondQ, CarreQ, TriangleQ,CreuxQ,PleinQ,PetitQ, GrandQ};
(* et puis aussi, plus tard:  pX1,pX2,pX3,pY1,pY2,pY3   *)

PleinQ[e4]:= True;
PleinQ[e5] := True;
PleinQ[e6]:= True;
PleinQ[e10] := True;
PleinQ[e11]:= True;
PleinQ[e12] := True;
PleinQ[_] :=False;

CreuxQ[e1] := True;
CreuxQ[e2] := True;
CreuxQ[e3] := True;
CreuxQ[e7] := True;
CreuxQ[e8] := True;
CreuxQ[e9] := True;
CreuxQ [_] := False;

GrandQ[e7]:= True;
GrandQ[e8] := True;
GrandQ[e9]:= True;
GrandQ[e10] := True;
GrandQ[e11]:= True;
GrandQ[e12] := True;
GrandQ[_] :=False;

PetitQ[e1] := True;
PetitQ[e2] := True;
PetitQ[e3] := True;
PetitQ[e4] := True;
PetitQ[e5] := True;
PetitQ[e6] := True;
PetitQ [_] := False;
```

```
CarreQ[e2] := True;
CarreQ[e5] := True;
CarreQ[e8] := True;
CarreQ[e11] := True;
CarreQ [_] := False;

RondQ[e1] := True;
RondQ[e4] := True;
RondQ[e7] := True;
RondQ[e10] := True;
RondQ[_] := False;

TriangleQ[e3] := True;
TriangleQ[e6] := True;
TriangleQ[e9] := True;
TriangleQ[e12] := True;
TriangleQ[_] := False;
ClearAll[Translate];
ClearAll[Dessine];
ClearAll[Translate2,tmp]
ClearAll[cFigure,CreerFigure,DansAlphabetQ,DansBureauQ, ObjectFrom];

Dessine [e1] :=  {Circle[{1/2,1/2},1/2]};
Dessine [e2] := {Line[{{0,0},{0,1},{1,1},{1,0},{0,0}}]};
Dessine [e3] :=  {Line[{{0,0},{1,0},{1/2,1},{0,0}}]};
Dessine [e4] :=  {Disk[{1/2,1/2},1/2]};
Dessine [e5] :=  {Rectangle[{0,0},{1,1}]};
Dessine [e6] :=  {Polygon[{{0,0},{1,0},{1/2,1},{0,0}}]};

Dessine [e7] :=  {Circle[{1,1},1]};
Dessine [e8] := {Line[{{0,0},{0,2},{2,2},{2,0},{0,0}}]};
Dessine [e9] :=  {Line[{{0,0},{2,0},{1,2},{0,0}}]};
Dessine [e10] :=  {Disk[{1,1},1]};
Dessine [e11] :=  {Rectangle[{0,0},{2,2}]};
Dessine [e12] :=  {Polygon[{{0,0},{2,0},{1,2},{0,0}}]};

Translate[v:{_,_}] [d_]:= d/.{Circle[p_,r__]:> Circle[p+v,r],
                                                                                          Disk[p_,r_]:>Disk[p+v,r],

      Rectangle[p1_,p2_] :> Rectangle[p1+v,p2+v],

      Polygon[l_List] :>  Polygon[Map[(#+v)&,l]],

      Line[l_List] :>  Line[Map[(#+v)&,l]]}

Translate2[o_,{n_}] :=Graphics [ Translate[{0,4n}][Dessine[o]]]
DansAlphabetQ[x_] := MemberQ[Univers,x];
DansBureauQ[x_] := MemberQ[Range[3], x];

CreerFigure[l:{{_?DansAlphabetQ,{_?DansBureauQ,_?DansBureauQ}}..}] :=
  cFigure[l]
CreerFigure[l_] :=  Error["ce n'est pas une fig",l]

ObjectFrom[f_cFigure] :=   Cases[f[[1]],  {x_,_} :> x]
Dessine[cFigure[l_]] := Map[Dessine,l]
Dessine[{x_?DansAlphabetQ,p_}] := Translate[2p][Dessine[x]]
```

```
ClearAll[AbscisseDe, OrdonneeDe]
AbscisseDe [f_cFigure][x_] :=
    Flatten[
        Cases[f[[1]],
            {x,{a_,b_}} :> a]
    ,1]
AbscisseDe[u_][x_] :=0

OrdonneeDe[f_cFigure][x_] :=
    Flatten[
        Cases[f[[1]],
            {x,{a_,b_}} :> b]
    ,1]
OrdonneeDe[u_][x_] :=0
ClearAll[LigneDe, ExtraitLignesDe]

LigneDe[e_] := Map[#[e]&,ListeProp]

ExtraitLignesDe[f_cFigure] :=Map[LigneDe,ObjectFrom[f]]
ExtraitLignesDe[u_] :=Map[LigneDe,u]

UniversLignes = ExtraitLignesDe[Univers];
ClearAll[PartageDeLignes, Qconnection]

PartageDeLignes[l1_, l2_] :=Count[MapThread[And,{l1,l2}], True]

Qconnection[q_Integer][l_, tab_] :=Select[tab, (PartageDeLignes[l,#] >= q)&]

ClearAll[Nsommets, Chemins, Qchemins,Piste,UpdatePiste,ChemSet,UpdateChemSet]

Nsommets[l_List] := Count[l,True]

Chemins[depart_List,arrivee_List, permis_List] /;(
      MemberQ[permis,depart]&&MemberQ[permis,arrivee]):=
    With[{q=Max[Nsommets[depart],Nsommets[arrivee]]},
        Qchemins[depart, arrivee, permis][q, 0, {depart}]
    ]

Chemins[depart_Symbol,arrivee_Symbol, permis_List] :=
  Chemins[LigneDe[depart],LigneDe[arrivee],permis]

Qchemins[depart_, arrivee_, permis_] [q_,n_,ens_]:=
    If[MemberQ[ens,arrivee],ChemSet[q,{Piste[arrivee]}],
        If[(n<=Length[permis]),(*
        on peut se restreindre aux chemins de lg <
          m en remplaçant Length[permis] *)
            (With[{
            ch= Qchemins[depart, arrivee, permis][q,n+1,
                Union@@Map[Qconnection[q][#, permis]&,ens]]},
        UpdateChemSet[ens,q,ch]]),
            If[(q <= 0),ChemSet[],
                Qchemins[depart, arrivee, permis][q-1,0,{depart}]]]]

UpdateChemSet[_,_,ChemSet[]]:=ChemSet[]
UpdateChemSet[s_,q_,ChemSet[q_,chems_]]:=
    With[{nchems=Union[Flatten[Outer[UpdatePiste[q],s,chems,1]]]},
        ChemSet[q,nchems]]
```

```
UpdateChemSet[_,_,r_ChemSet]:=r

UpdatePiste[q_Integer][x_,Piste[y_,l___]]:=
    If[PartageDeLignes[x,y]≥q,{Piste[x,y,l]},{}]
ClearAll[Nom,Voir,VoirDefElem]

Nom[x_List] :=
With[{debut = Take[x,7]},
With[{pos=Position[UniversLignes,debut]},
            If[pos==={},"Unknown"@@debut,Univers[[pos[[1,1]] ]] ]]]

Voir[ChemSet[q_,l_]]:=
        (Print["chemin de connexite ",q,":"];Print["\t\t\t",Voir/@l // TableForm])

Voir[c_] :=  c/.  {
            Piste[l___] :> Piste@@Nom/@{l},
            DefElem[l0_,l1_] :>  VoirDefElem[l0,l1],
            lb:{(True|False)..} :> Nom[lb]
            }

VoirDefElem[a_,b_] :=
        With[{change = MapThread[Xor,{a,b}]},
            With[{ posChange = Position[change,True]},
                With[{ un2zero= Select[posChange,a[[First[#]]]&],
                                  zero2un = Select[posChange, !a[[First[#]]]&]},
                "D-elem"@@
          Join[Map[Rule[ListeProp[[ First[#] ]],0]&,un2zero],
            Map[Rule[ListeProp[[ First[#] ]],1]&,zero2un]]]]]
ClearAll[Appariements,App1,App2,
                                    AssocSet,Paire,FromTo,
                                  UpdateAssocSet,Regroupe]

Appariements[l1_List, l2_List, fct_:App2] := fct[l1,l2]

App1[l1_List, l2_List]:=Regroupe[l1]/@(App1[l1,l2,{AssocSet[]}])

App1[l1_List,{},r_List]:=r
App1[l1_List,{x_,l___},r_List]:=
    App1[l1,{l},Flatten[UpdateAssocSet[r,x]/@l1,1]]

UpdateAssocSet[r_List, y_][x_]:=Map[Join[#,AssocSet[FromTo[x,y]]]&,r]

Regroupe[l1_List][s_AssocSet]:=
  AssocSet@@Map[FromTo[#,Cases[s,FromTo[#,y_]:>y]]&,l1]

App2[l1_List,l2_List] := {AssocSet@@
            Which [
                (Length[l1] === Length[l2]), MapThread[FromTo[#1,{#2}]&,{l1,l2}],
                (Length[l1] > Length[l2]),
        Join[MapThread[FromTo[#1,{#2}]&,{Take[l1,Length[l2]],l2}],
          FromTo[#,{}]&/@Drop[l1,Length[l2]]],
                (Length[l1] < Length[l2]),Join[
                    MapThread[FromTo[#1,{#2}]&,{Drop[l1,-1],Take[l2,Length[l1]-1]}],
                 {FromTo[Last[l1],Drop[l2,Length[l1]-1] ]}]]}
ClearAll[DefElem,CurrentDefElem,
     DefElem1, DefElem2,DefElem3,DefElem4,DefElem5]

CurrentDefElem = DefElem1;
```

```
DefElem[a_,b_][argu_List] /;VectorQ[argu]:= CurrentDefElem[a,b][argu]
DefElem[a_,b_][argu_List]:= (CurrentDefElem[a,b][#])& /@ argu

DefElem1[a_,b_][argu_List] :=
        With[{change = MapThread[Xor,{a,b}]},
            With[{ posChange = Position[change,True]},
                With[{ un2zero= Select[posChange,a[[First[#]]]&],
          zero2un = Select[posChange, !a[[First[#]]]&]},
                ReplacePart[ReplacePart[argu,False,un2zero],True,zero2un]  ]]]

DefElem2[a_,b_][argu_List] := b

DefElem3[a_,b_][argu_List] := If[argu===a,b,argu]

DefElem4[a_,b_][argu_List] :=
        With[{change = MapThread[Xor,{a,b}],
                         domaine=MapThread[(!Xor[#1,#2])&,{a,argu}]},
        With[{inter=MapThread[And,{change,domaine}]},
            With[{ posChange = Position[inter,True]},
                With[{ un2zero= Select[posChange,a[[First[#]]]&],
                                zero2un = Select[posChange, !a[[First[#]]]&]},
                        ReplacePart[ReplacePart[argu,False,un2zero],True,zero2un]  ]]]]

(* Avec connaisance a priori codé dans DefElem *)
DefElem5[a_,b_][argu_List]:=
    With[{changeForm=Or@@MapThread[Xor,{Take[a,{1,3}],Take[b,{1,3}]}],
                            changeTaille=Or@@MapThread[Xor,{Take[a,{6,7}],Take[b,{6,7}]}],
                         changeCouleur=Or@@MapThread[Xor,{Take[a,{4,5}],Take[b,{4,5}]}]},
        Join[
            If[changeForm && (Take[argu,{1,3}] === Take[a,{1,3}]),Take[b,{1,3}],
        Take[argu,{1,3}]],
            If[changeCouleur && (Take[argu,{4,5}] === Take[a,{4,5}]),Take[b,{4,5}],
        Take[argu,{4,5}]],
            If[changeTaille && (Take[argu,{6,7}] === Take[a,{6,7}]),Take[b,{6,7}],
        Take[argu,{6,7}]]
        ]]
ClearAll[PisteMin , Deformation,Successivement,Duplique,Laisse]

Deformation[x_,{},_]:=Laisse[x]
Deformation[x_,{y_},permis_List]:=Successivement@@PisteMin[x,y,permis]
Deformation[x_,l_List,permis_List]/;(Length[l]>1):=
  Duplique@@(Deformation[x,{#},permis]&/@l)
```

```
PisteMin[l1_, l2_,permis_List] :=
  With[{ch = Chemins[l1,l2,permis]}, First[Sort[ch[[2]] ]]] endMinput
      ClearAll[Transformation]

      Transformation[Successivement[x_]]:=Successivement[DefElem[x,x]]
      Transformation[s_Successivement]:=(DefElem@@#)&/@Partition[s,2,1]
      Transformation[x_Laisse]:=x

      Transformation[l_]:=Transformation/@l
      ClearAll[Applique]

      (* il se peut qu'il n'y ait pas d'argument sur lequel appliquer la
          transformation *)
      Applique[f_,{}]:={}

      Applique[Successivement[f_],argument_List]:=f[argument]
      Applique[Successivement[f_,suite__],argument_List]:=
          Applique[Successivement[suite],f[argument]]

      Applique[d_Duplique, argument_List]:=
        Join@@((Applique[#,argument]&)/@(List@@d))

      Applique[Laisse[_],argument_List]:={}
      ClearAll[Resolve,Choisir,Simultanement,AppApp1,AppApp2]

      AppApp1[l1_List,l2_List]:={Choisir[First[l1],First[l2]]}
      AppApp2[l1_List,l2_List]:=Flatten[Outer[Choisir,l1,l2],1]

      Resolve[A_List,B_List,C_List,
                                  univers_List:UniversLignes,fctApp_:App2,fctAppApp_:AppApp1]:=
          With[{aa=Range[Length[A]],bb=Range[Length[B]],cc=Range[Length[C]]},
          With[{appAB=Appariements[aa,bb,fctApp],appAC=Appariements[aa,cc,fctApp]},
          With[{appABC=fctAppApp[appAB,appAC]},
                      Print["appariement A-B : ", appAB];
                      Print["appariement A-C : ", appAB];
                      Print["appariement des appariements :  ", appABC];
          With[{paireDeList=
                    appABC/.Choisir[a_AssocSet,b_AssocSet]:>
                        Simultanement@@Transpose[{List@@a,List@@b}]},
          With[{trans=paireDeList /.  {FromTo[idx_,l1_],FromTo[idx_,l2_]}:>
                                  Domaine[idx,
                                                                  Transformation[Deformation[A[[idx]],Map[B[[#]]&,l1],univers]],

                                                                  Map[C[[#]]&,l2]]},
                  Print["transformation :",trans//Voir];
          With[{allSolutions=Choisir@@(trans/.Domaine[_,f_,a_]:>
                                                                                                  Applique[f,a]
      /.Simultanement:>Join)},
          With[{solutions=Union[List@@allSolutions]},
          With[{perfsol=Map[Paire[Count[allSolutions,#], #]&,solutions]},
          With[{bestsol=Sort[perfsol,(#1[[1]]<#2[[1]])&]},
                  Print["solutions (avec multiplicité) = ",bestsol//Voir];
                                    Choisir@@((#[[2]])&/@bestsol)
          ]]]]]]]]
```

```
ClearAll[Resoud,VoirSol]

Resoud[A_List,B_List,C_List,
                                univers_List:UniversLignes,fctApp_:App2,fctAppApp_:AppApp1]:=
    With[{sol=List@@Resolve[A,B,C,univers,fctApp,fctAppApp]},
        Print["A:"];Show[VoirSol[A]];
        Print["B:"];Show[VoirSol[B]];
        Print["::C:"];Show[VoirSol[C]];
        Print["D:"];Show[GraphicsArray[VoirSol/@sol]];
        sol]

VoirSol[s_]:=MapIndexed[Translate2,Nom/@s]
```

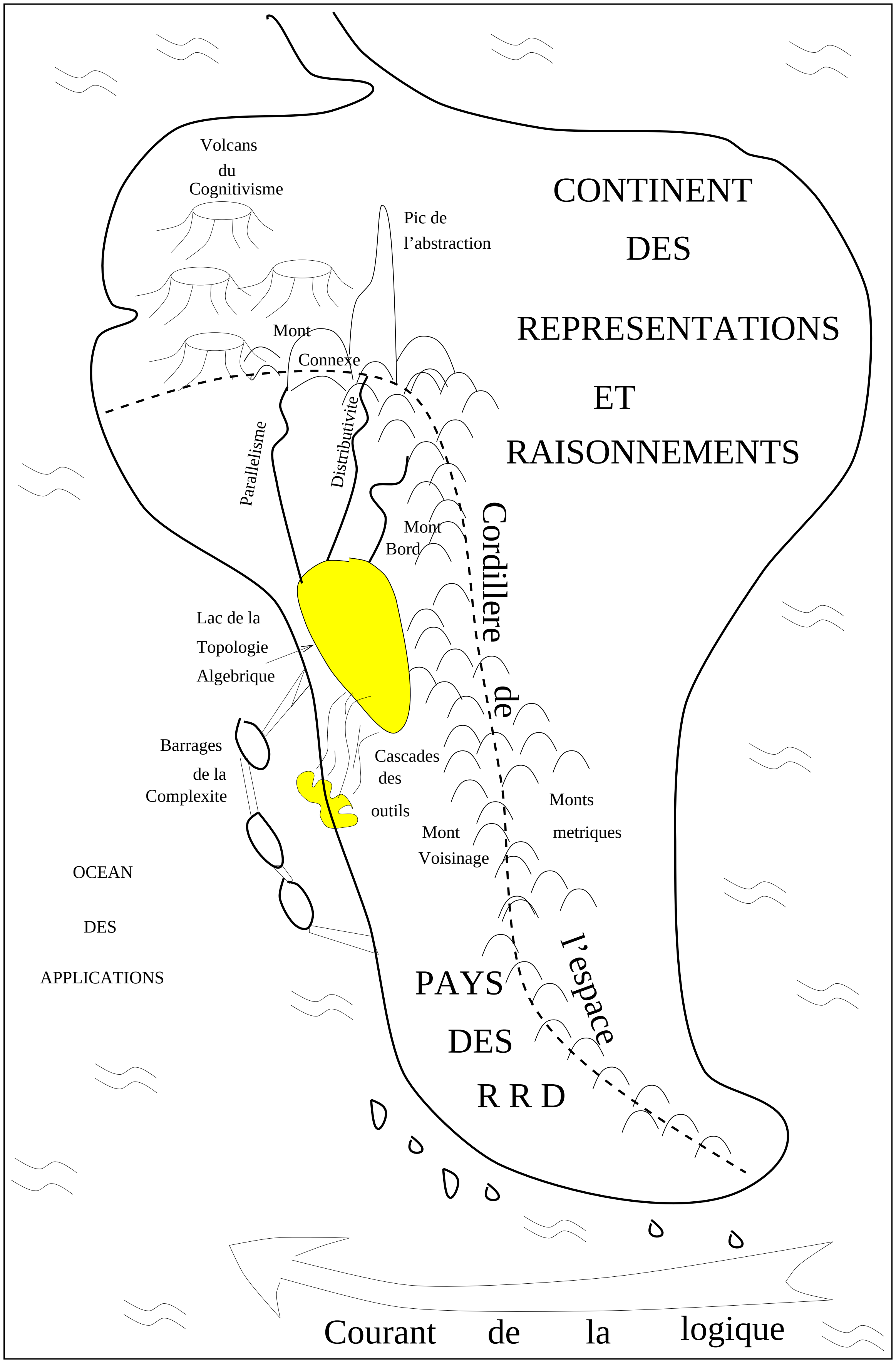


FIG. **C.1** – Le continent des Représentations et des Raisonnements est traversé par la Cordillère de l'Espace, qui forme la frontière naturelle du pays des RRD. Sur le côté de la Cordillère qui est dans le pays des RRD, se trouvent les Monts Bord, Voisinage et Connexe, tandis que de l'autre côté se trouvent les Monts Métriques. Les eaux des fleuves Parallèlisme et Distributivité rejoignent celles du Lac de la Topologie Algébrique, avant de se précipiter dans les Cascades des Outils Effectifs. Cependant, elles doivent d'abord franchir les Barrages de la Complexité avant de pouvoir se jeter dans l'Océan des Applications.

Index

Bibliographie

[ABY85] John R. Anderson, C. Franklin Boyle, and Gregg Yost. The geometry tutor. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 1–7, Los Angeles, CA, August 1985. Morgan Kaufmann.

[AIS+77] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A pattern language.* Oxford University Press, 1977.

[Ale82] P. Alexandroff. *Elementary concepts of topology.* Dover publications, New-York, 1982.

[Arn69] Rudolh Arnheim. *Visual Thinking.* University of California Press, 1969. trad. française : "La pensée visuelle", Champs-Flammarion, 1976.

[Atk74] R. Atkin. *Mathematical Structure in Human Affairs.* Heinemann, 1974.

[Atk76] R. Atkin & al. Fred CHAMP positional chess analyst. *International Journal of Man-Machine Studies*, 8:517–529, 1976.

[Atk77] R. H. Atkin. *Combinatorial Connectivities in Social Systems.* Verlag, 1977.

[Atk81] R. Atkin. *Multidimensional Man.* Penguin, 1981.

[Ber97] P. Bernat. Représenter et manipuler des connaissances dans un environnement d'apprentissage de r'esolution de problèmes. *Revue d'intelligence artificielle*, 11(2):213–238, 1997.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern oriented software architecture – A system of patterns.* Wiley, 1996.

[Cas82] John Casti. Topological methods for social and behavioral systems. *J. of General System*, 8:187–210, 1982.

[Cas92] John Casti. *Reality Rules: Picturing the world in mathematics. II — The Frontiers.* Wiley, 1992.

[Cas94] J. L. Casti. *Complexification.* Harper Perennial, 1 edition, 1994.

[Cla81] B. L. Clarke. A calculus of individuals based on connection. *Notre Dame Journal Of Formal Logic*, 23(3):204–218, 1981.

[CQ69] Allen Collins and Ross Quillian. Retrieval time from semantic memoy. *J. of verbal learning and verbal behavior*, 9:240–247, 1969.

[DE93] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes. Technical Report UIUCDCS-R-93-1787, Univ. of Illinois at Urbana Champaign, january 1993.

[Den92] M. Denis. Modèles mentaux et imagerie mentale. In *Les modèles mentaux, approche cognitive des représentations*, chapter 3, pages 79–98. masson, Paris, 1992.

[Die86] J. Dieudonné. *Abrégé d'histoire des Mathématiques.* Hermann, Paris, 2 edition, 86.

[Dor86] P. Doreian. Analysing overlaps in food webs. *J. Soc. & Biol. Structures*, 9:115–139, 1986.

[DS96] Tamal K. Dey and Guha Sumanta. Algorithms for manifolds and simplicial complexes in euclidean 3-space (preliminary version). In *STOC'96*, pages 398–407, Philadelphia PA, USA, 1996. ACM.

[Elt94] H. Elter. *Etude de structures combinatoires pour la représentation de complexes cellulaires.* Thèse de doctorat, Université Louis Pasteur de Strasbourg, Septembre 1994.

[Eva68] T.G. Evans. *Semantic information processing*, chapter A program for the solution of a class of geometric analogy intelligence test questions. MIT Press, 1968.

[Fah79] Scott Fahlman. *NETL : a system for representing and using real-word knowledge.* MIT Press, 1979.

[Gar97] H. Gardner. *Les formes de l'intelligence.* Éditions Odile Jacob, Paris, April 1997.

[GHJV96] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison Wesley, Reading, 1996.

[Gin97] Marie-Dominique Gineste. *Analogie et cognition.* Psychologie et Sciences de la pensée. PUF, février 1997.

[GJ92] E. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, August 1992. Springer-Verlag.

[God71] C. Godbillon. *Élément de topologie algébrique.* Hermann, 1971.

[Got94] N. M. Gotts. How far can we 'C'? defining a 'doughnut' using connection alone. In Pietro Torasso Jon Doyle, Erik Sandewall, editor, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 246–257, Bonn, FRG, May 1994. Morgan Kaufmann.

[Gou80] P. Gould. Q-analysis, or a language of structure : An introduction for social scientists, geographers and planners. *International Journal of Man-Machine Studies*, 13:169–199, 1980.

[Gun94] Jeremy Gunawardena. Homotopy and concurrency. *Bulletin of the EATCS*, 1994.

[Ha91] J. P. Haton and al. *Le raisonnement en intelligence artificielle.* InterEditions, 1991.

[Hen79] Garry Hendrix. *Associative Networks: representation and use of knowledge by computers*, chapter Enconding knowledge in partitionned networks, pages 51–92. Findler Academic Press, New York, 1979.

[Hen94] M. Henle. *A combinatorial introduction to topology.* Dover publications, New-York, 1994.

[HHNT86] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction – Processes of Inference, learning, and Discovery.* The MIT Press, 1986.

[HR95] M. Herlihy and S. Rajsbaum. *Computer Science Today - Recent Trends and Developments*, chapter Algebraic Topology and Distributed Computing: A Primer, pages 203–217. Number 1000 in Lecture Notes in Computer Sciences. Springer-Verlag, 1995.

[HY88] J. G. Hocking and G.S. Young. *Topology.* Dover publications, New-York, 1988.

[JL92] N. Johnson-Laird. La théorie des modèles mentaux. In *Les modèles mentaux, approche cognitive des représentations*, chapter Introduction, pages 1–21. masson, Paris, 1992.

[Joh90] J. Johnson. Gradient polygons: Fundamentals primitives in hierachical computer vision. In *Proc. Symp. in honor of Professor J.-C. Simon*, Paris, October 1990. AFCET.

[Joh91a] J. Johnson. *The mathematical revolution inspired by computing*, chapter The mathematics of complex systems, Johnson J. and Loomes M. eds., pages 165–186. Oxford University Press, 1991.

[Joh91b] J. Johnson. *Transport Planning and Control*, chapter The dynamics of large complex road systems, Griffiths J. ed., pages 165–186. Oxford University Press, 1991.

[Lak84] Imre Lakatos. *Preuves et Réfutations*. Hermann, 1984.

[Nie93] Oscar Nierstrasz. Composing active objects. In P. Wegner G. Agha and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 151–171. MIT Press, 1993.

[PLC91] J. Pimm, J. Lawton, and J. Cohen. Food web patterns and their consequence. *Nature*, 350:669–674, 25 April 1991.

[Pra91] Vaughn Pratt. Modeling concurrency with geometry. In ACM-SIGPLAN ACM-SIGACT, editor, *Conference Record of the 18th Annual ACM Symposium on Principles of Programming Languages (POPL '91)*, pages 311–322, Orlando, FL, USA, January 1991. ACM Press.

[Ros95] Jean-Pierre Rossi. Les représentations. Polycopié de cours pour le dea de sciences cognitives d'orsay, LIMSI, U. Paris-Sud, 1995.

[RS97] F. Reinhardt and H. Soeder. *Atlas des mathématiques*. Encyclopédies d'aujourd'hui. Le Livre de Poche, Paris, 1997.

[SA77] Roger Schank and Robert Abelson. *Scripts, Plans, Goals and understanding*. Hillsdale, 1977.

[Sab90] Gérard Sabah. *L'intelligence artificielle et le langage, la représentation des connaissances*. Hermès, Paris, 1990.

[SM81] E. E. Smith and D. L. Medin. *Categories and concepts*. Harvard University Press, 1981.

[Sow84] J. F. Sowa. *Conceptual structures*. Addison-Wesley, 1984.

[Tau93] A. Taurisson. *Pensée mathématique et gestion mentale, pour une pédagogie de l'intuition mentale*. Bayard Editions, 1993.

[THNG90] P. Thagard, K. J. Holyak, G. Nelson, and D. Gochfeld. Analog retrieval by constraint satisfaction. *Artificial Intelligence*, 46(3):259–310, December 1990.

[Thu38] L. L. Thurstone. Primary mental abilities. *Psychometric monographs*, 1, 1938.

[vG91] R. van Glabbeek. Bisimulations for higher dimensional automata. Manuscript available electronically as http://theory.stanford.edu/~rvg/hda, June 1991.

[Wit21] Ludwig Wittgenstein. *Tractatus logico philosophicus*. Gallimard, 1921.

[WS90] S. H. Weber and A. Stolcke. l_0 : A testbed for miniature language acquisition. *International Computer Science Institute*, 1990.

[Yvo96] F. Yvon. *Prononcer par analogie : motivation, formalisation et évaluation*. PhD thesis, E.N.S.T., Paris, May 1996.

Erika Valencia, 29 Août 1997, *Version 1.1*